SOFTWARE INTEROPERABILITY: Issues at the Intersection between Intellectual Property and Competition Policy

Università degli Studi di Torino

SOFTWARE INTEROPERABILITY
Issues at the Intersection between Intellectual Property and Competition Policy

Federico Morando

Dottorato di Ricerca in Analisi Comparata dell'Economia,

del Diritto e delle Istituzioni

Codice del Settore Scientifico Disciplinare (SSD): IUS/02

Comparative Analysis of Institutions, Economics and Law (IEL)

Università degli Studi di Torino

SOFTWARE INTEROPERABILITY
Issues at the Intersection between Intellectual Property and Competition Policy

Federico Morando

Dottorato di Ricerca in Analisi Comparata dell'Economia, del Diritto e delle Istituzioni (IEL)

Comparative Analysis of Institutions, Economics and Law (IEL)

Ciclo XXI

Supervisors: Prof. Ben Depoorter

Prof. Marco Ricolfi

Candidate: Federico Morando

Programme Coordinator: Prof. Giovanni B. Ramello

Dipartimento di Economia “S. Cognetti de Martiis”
Dipartimento di Scienze Economiche e Finanziarie “G. Prato”
Dipartimento di Scienze Giuridiche

Universiteit Gent

SOFTWARE INTEROPERABILITY

Issues at the Intersection between Intellectual Property and Competition Policy

Federico Morando

Doctoraat in de Rechten

Thesis in Cotutelle

a Valentina

ACKNOWLEDGMENT

Several people contributed, directly or indirectly, to the realization of this Ph.D. thesis. First of all, I am grateful to my supervisors, Prof. Ben Depoorter and Prof. Marco Ricolfi. I expected to find in them competent scholars and capable teachers, but I have been glad to find also excellent mentors and good friends. Moreover, I have been lucky enough to be guided by two scholars with different and frequently complementary approaches to investigation and analysis and to the academic world in general and this dramatically enriched my research experience. I would also like to express my thanks to Prof. Gianmaria Ajani and, with him, all the people who believed in the possibility of creating the IEL Ph.D., a doctoral course that I really enjoyed from any point of view and which allowed me to pursue interdisciplinary studies in an international context. I am confident that these and new committed people, now under the co-ordination of Prof. Giovanni B. Ramello, will be able to further improve the quality of this programme. Let me also acknowledge, in particular, the contribution to my thesis coming from the comments of Prof. Pierre Jean Benghozi and Prof. Pierre Garello. Furthermore, I would like to thank Prof. Boudewijn Bouckaert, the other Professors and the entire staff of the Centre for Advanced Studies in Law and Economics at the Law School of the Univesiteit Van Gent: I am especially grateful to Ms. Katelijne Verstichel for her valuable help in solving the myriad of small bureaucratic issues related to a thesis in cotutelle.

This thesis also owes a lot to the members of two other academic groups. I would like to express my deep appreciation to my friends and colleagues at Bocconi University in Milan, for many fruitful discussions about the topics of this dissertation and about intellectual property and competition policy in general. I would like to mention, among them, Prof. Maria Lillà Montagnani, Prof. Paola Magnani and Prof. Federico Ghezzi. Moreover, I thank all the fellows of the NEXA Center for Internet & Society of the Politecnico di Torino and, in particular, Prof. Juan Carlos De Martin, co-director of the Center together with Prof. Marco Ricolfi. This wonderful group of enthusiast and committed scholars, researchers and practitioners, coming from various disciplines (including engineering, law, economics and management) provided me with several specific insights for my thesis and, more generally, convinced me of the value of a multidisciplinary approach, backed by a rigorous methodology.

In addition, I am grateful to all my Ph.D fellows. Particularly, I wish to thank Enrico Bertacchini, with whom I spent many days in Ghent (and quite some evenings talking about law & economics in front of a Belgian beer), Branko Radulovic and Elisa Vecchione.

The warmest thanks also go to my parents, who encouraged me to pursue my studies and always supported me.

Last, but not least, I thank my wife Valentina for having made me the happiest man on Earth during the years of this dissertation and – I hope – during the rest of our lives.

PAPER 1 - TABLE OF CONTENTS

PAPER 2 - TABLE OF CONTENTS

PAPER 3 – TABLE OF CONTENTS

General introduction

1. General plan of the dissertation

The dissertation project proceeds through three papers, analyzing issues related to software interoperability and respectively pertaining to one of the three following interdependent levels of analysis.

The first level addresses the legal status of software interoperability information under current intellectual property law (focusing on copyright law, which is the main legal tool for the protection of these pieces of code), trying to clarify if, how and to what extent theses pieces of code (and the associated pieces of information) are protected *erga omnes* by the law.

The second level complements the first one, analyzing legal and economic issues related to the technical possibility of actually accessing this interoperability information through reverse engineering (and software decompilation in particular). Once a *de facto* standard gains the favor of the market, reverse engineering is the main self-help tool available to competitors in order to achieve interoperability and compete "inside this standard".

The third step consists in recognizing that – in a limited number of cases, but which are potentially of great economic relevance – market failures could arise, despite any care taken in devising checks and balances in the legal setting concerning both the legal status of interoperability information and the legal rules governing software reverse engineering. When this is the case, some undertakings may stably gain a dominant position in software markets, and possibly abuse it. Hence, at this level of analysis, competition policy intervention is taken into account.

1.1. Summary of the three papers

The first paper of the present dissertation shows that interoperability *specifications* are not protected by copyright.[1] In the paper, I argue that existing doubts and uncertainty are typically related to a poor understanding of the technical nature of software interfaces. To remedy such misunderstanding, the paper focuses on the distinction between interface *specifications* and *implementations* and stresses the difference between the steps needed to access to the ideas and principle constituting an interfaces specification and the re-implementation of a functionally equivalent interface through new software code. At the normative level, the paper shows that no major modifications to the existing model of legal protection of software (and software interfaces) are needed; however, it suggests that policymakers could reduce the Fear of legal actions, other forms of legal Uncertainty and several residual Doubts (FUD) by explicitly stating that interface specifications are unprotectable and freely appropriable.

In the second paper, I offer a critique of legal restraints on software reverse engineering, focusing in particular on Europe, but considering also similar restraints in the US, in particular in the context of the Digital Millennium Copyright Act. Through an analysis of entry conditions for late comers and of the comparative costs of developing programs in the first place or reverse engineering them, the paper shows that limitations on decompilation imposed by article 6 of the Software Directive were mostly superfluous and basically non-binding at the time of drafting.[2] What is more, the paper shows that nowadays new – and largely unanticipated – developments in software development models (e.g. open source) make these restraints an obstacle to competition against dominant incumbent controlling software platforms. In fact, limitations on the freedom to decompile obstacle major reverse engineering projects performed in a decentralized way, as in the context of an open source community. Hence, since open source projects are the most credible tools to recreate some competitive pressure in a number of crucial software markets, the paper recommends creating a simpler and clear-cut safe harbor for software reverse engineering.

---

[1] This result is specifically discussed with respect to the legal systems of the economically more important areas of the world: EU, US and Japan. The analysis is also likely to apply (with minor adaptations) to any copyright system compliant with the Berne Convention (and hence to all the member of the WTO, since the Convention has been incorporated in the TRIPs agreement), since it is mainly based on general copyright law principles, technical arguments and economic reasoning.

[2] To be sure, I do not claim that including these limitations in the Directive have been irrational: in fact – at the time of the drafting – they were essentially innocuous from an economic point of view, but they were "diplomatically" very useful. In fact, they have been inserted mainly as part of a complex political balancing exercise, taking into account the interests of two organized lobbying groups, representing the interest of the US big software houses (on one side) and of the European informatics industry (on the other).

The third paper claims that, in software markets, refusal-to-deal (or "information-withholding") strategies are normally complementary with tying (or "predatory-innovation") strategies, and that this complementarity is so relevant that dominant platform controllers need to couple both in order to create significant anti-competitive effects. Hence, the paper argues that mandatory unbundling (i.e. mandating a certain degree of modularity in software development) could be an appropriate – and frequently preferable – alternative to mandatory disclosure of interoperability information. However, considering the critiques moved from part of the literature to the Commission's Decision in the recent European Microsoft antitrust case, an objection to the previous argument could be that – also in the case of mandatory unbundling – one should still determine the minimum price for the unbundled product. The last part of the paper applies some intuitions coming from the literature concerning complementary oligopoly[3] to demonstrate that this objection is not well grounded and that – in software markets – mandatory unbundling (modularity) may be a useful policy even if the only constraint on the price of the unbundled good is the one of non-negativity.

1.2. An overall background

In the background to the three papers forming the dissertation at hand, there is a quite precise understating of the economics of software markets and of the legal tools devised to stimulate innovation in these markets. In fact, my view of these fields owes a lot to several lawyer-economists, in particular to the writings of Reichman[4] and Weiser,[5] but also Samuelson, Scotchmer and others,[6] and to the industrial organization literature, in particular to the contributions of Prof. Farrell, Prof. Tirole and Prof. Rochet.[7] However, no single work can easily be used as a simple reference summarizing this general understanding of the law & economics of the software industry; therefore, I will use this introduction to provide some hints, shaping the background of the following three papers.

There are no especially original contributions in the following pages of this *General Introduction*: indeed, I decided to provide this preface to the dissertation precisely in order to be able to focus more each individual paper on the issues, which I considered to constitute my actual modest contribution to the literature. Readers with some background in the law & economics of the software industry will likely be able to figure out my understanding of the economics of the software industry just by reading each paper, hence, they are advised of the fact that they may safely skip the following part of this introduction or quickly pass through it.

2. A Foreword about the economics of software markets

In each of the papers making up this dissertation, I will frequently mention network effects, switching costs and other economic phenomena. This paragraph briefly summarizes these concepts, referring to some relevant contributions in these fields.

2.1. Cost structure

It is well known that, in the software industry, fixed (research & development) costs are potentially very high, while marginal costs are almost negligible, because of the well-known characteristics of digital information. A predominant part of these fixed costs is also sunk, in the sense that intellectual assets generated by these investments are highly specific to a certain software project and may be hardly (if at all)

---

[3] See, in particular, FRANCESCO PARISI, et al., *Duality in Property: Commons and Anticommons*, 25 International Review of Law and Economics, 578--591 (2005); FRANCESCO PARISI & BEN DEPOORTER, *The Market for Intellectual Property: The Case of Complementary Oligopoly*, *in* The Economics of Copyright: Developments in Research and Analysis, (W. Gordon & R. Watt eds., 2003); GIUSEPPE DARI-MATTIACCI & FRANCESCO PARISI, *Substituting Complements*, 2 Journal of Competition Law and Economics, 333--347 (2006); STEVEN C. SALOP, *Competition and Integration Among Complements, and Network Market Structure*, 40 The Journal of Industrial Economics, 105--123 (1992).

[4] In particular, J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Columbia Law Review, 2432 (1994).

[5] P. J. WEISER, *Law And Information Platforms*, 1 J. Telecomm. & High Tech. L., 1 (2002) and in particular P. J. WEISER, *The Internet, Innovation, and Intellectual Property Policy*, 103 Columbia Law Review, 534--613 (2003).

[6] PAMELA SAMUELSON, *Benson Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions*, 39 Emory L.J., 1025 (1990); PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, 94 Columbia Law Review, 2308--2431 (1994); PAMULE SAMUELSON & S. SCOTCHMER, *The Law and Economics of Reverse Engineering*, 111 Yale Law Journal, 1575--1663 (2002).

[7] See, in particular: JOSEPH FARRELL & PHILIP J. WEISER, *Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age*, 17 Harvard Journal of Law & Technology, 85 (2003); J. C. ROCHET & J. TIROLE, *Platform Competition in Two-Sided Markets*, 1 Journal of the European Economic Association, 990--1029 (2003); J. C. ROCHET & J. TIROLE, *Two-Sided Markets: A Progress Report*, 37 RAND Journal of Economics, 645--667 (2006) (or J. C. ROCHET & J. TIROLE, *Two-Sided Markets: An Overview*, IDEI Toulouse working paper (March, 2004)). As a general reference, see also JEAN TIROLE, *The Theory of Industrial Organization*, (MIT Press, Cambridge: Mass. 1988).

resold or otherwise "recycled" in different projects. To be sure, the zero-marginal-cost peculiarity of industries based on digital goods (but similar concepts apply to biotechnologies) should be taken into account while analyzing competition in these markets. Nevertheless, it is also important to be aware of the fact that the characterization of "zero-marginal-cost economics" (sometimes used to describe the peculiarities of digital and bio technologies) may be highly misleading (an is probably based on a simplistic perfect competition model, in which price is equal to marginal cost and hence dropping to zero). In fact, a null marginal cost is not necessarily crucial in determining whether undertakings may recoup their sunk costs, unless one is able to show a model in which the profit mark-up must necessarily be a multiple of the marginal cost. Indeed, if competition is perfect – no matter whether the marginal cost is zero or any "c" – sunk costs cannot be recouped. Entry conditions are the real problem, potentially leading to market failures. In fact, if the difference between the sunk costs of the first entrant and those of followers is positive – as may happen in these markets – we are in a "negative barriers to entry" environment.[8] If barriers to entry are negative, late comers can compete in a stronger and stronger way, driving profits towards zero and hence discouraging innovation and entry in the first place. (That happens since the first entrant looks at the industry *ex ante* and tries to solve its future "competitive game" by backward induction, understanding that profit prospects are inexistent and is likely to decide not to enter at all). Therefore, a better name than "zero marginal cost economics" could be "free duplication of sunk investments economics" or "negative barriers to entry economics". Indeed, the reason for which some kind of artificial barrier to entry is highly needed in software related markets is that each copy of a good – coupled with a cheap general-purpose computer – is a perfect mould, that can be used to "clone" identical goods. The point is not the marginal cost, but the fact that each product is also a production asset with no capacity constraints. Hence, the need for intellectual property rights or other exclusion devices (typically including trade secret and possibly including various other liability systems). To be sure, the level of marginal costs is hardly relevant: the point is that productive capacity may be generated by late comers without incurring any sunk costs, unless there are intellectual property rights (or other legal or technical tools) preventing this kind of free riding. Hence, as long as there is a high up-front entry cost (more or less similar for each and every competitor), new entrants will be deterred when the degree of competition is high enough not to permit them to recoup their initial investment. And that means – as long as there are no major coordination problems or discontinuities in profits – that it is possible to recoup the initial investment (again, no matter how high or low marginal costs are).[9]

2.2. Systems and networks

In general, the term "network effect" refers to the change in the perceived value of a product, caused by a change in the number of people who use it (the network of users). The term "network externality" is reserved by a few authors to network effects that lead to a market failure: I will not adopt this convention and the two expressions are synonyms in this dissertation.[10] Following one of the first definitions of "network effect", provided by Katz and Shapiro in 1985, we are in the presence of network externalities if "the utility that a user derives from consumption of a good *increases* with the number of other agents consuming the good".[11] The focus on positive network effects is quite interesting: in fact, the more people use a product, the more its value increases and the producer gains market power, even without any other improvements to the product (that is, without technical improvements, innovation or investments by the producer). For this reason, network effects (especially if coupled with learning effects) have been associated with natural monopoly, essential facilities, standardization and tipping.

---

[8] According to the definition of DENNIS W. CARLTON & JEFFREY M. PERLOFF, *Modern Industrial Organization*, (Denise Clinton ed., Addison-Wesley Third ed, Reading, Massachusetts. 2000), pag. 76—77: "Because long run profits can only persist if a firm has an advantage over potential entrants, a logical definition of a *long-run barrier to entry* is a cost that must be incurred by a new entrant that incumbents do not (or have not had to) bear." (This definition is an adaptation from the one originally proposed from Stigler: GEORGE STIGLER, *Barriers to Entry, Economies of Scale, and Firm Size*, in George Stigler, *The Organization of Industry*, (Homewood, IL, Richard D. Irvin, 1968).) *See* also TIROLE, *The Theory of IO*, p 305.

[9] To be sure, the recoupment of sunk costs is only *possible*, not guaranteed, and it depends on the market response; the relevant point is that, as long as new entrants have up-front sunk costs which are similar to those of the incumbent, they will not enter, unless the incumbent is also able to recoup its sunk costs (because these costs are not yet "sunk" for a potential entrant, looking at the market from outside).

[10] In presence of network effects, a consumer consumption decision modifies the welfare of other consumers without changing the price faced by the consumer under examination: this is the definition of externality, even without complete market failure.

[11] Emphasis added. MICHAEL L. KATZ & CARL SHAPIRO, *Network Externalities, Competition and Compatibility*, 75 American Economic Review, 424 (1985).

As explained by several authors,[12] despite the fact that network effects are generally defined in a direct sense (with individual utility increasing with the total number of users), the "transmission mechanism" of this increase is frequently indirect: "a good becomes more valuable as more consumers use it because there is a greater variety of a complementary goods available".[13] This is the case in software markets and the operating systems market is especially concerned by indirect network effects: network effects are at the base of the so called "applications barrier to entry".[14]

Learning effects are another example of demand-side phenomenon, for which the utility that consumers derive from a durable good rises over time, typically as an effect of improved skills in using this good. These improved skills are the effect of a (more or less conscious) investment in learning: this investment is usually sunk, in the sense that it is (at least partially) specific to a certain model or brand of good and may be lost when switching to another good, even if the two were perceived as fungible from an ex ante perspective (with respect to the learning investment).

To be sure, network effects and learning costs are a *per se* positive phenomenon, which increase the utility consumers derive from the consumption of a good. If the number of users increases, *ceteris paribus*, the demanded quantity will increase and the demand curve will experiment an upward shift: the consumer surplus increases because the willingness to pay increases with the perceived value of the product. In fact, as economies of scale and experience, network effects can even "justify" (in term of social welfare) the existence of a monopoly: in non-technical literature, this phenomenon is frequently described as "the benefit of standardization". However, competition may feel the effects of these phenomena, because the utility derived from the product of a certain producer may be lost when consumers shift to another producer, hence increasing switching costs. These costs essentially consist of all the expenditures (and other inconveniences) related to changing one's suppliers.[15] Some of these costs include "the need to incur specific capital investment or the loss in current output in order to switch to alternative inputs, […] specific investment in production process, learning and human capital investment, retooling costs or other investments, uncertainty about quality and reputation of unknown suppliers".[16] Clearly, the higher these costs, the higher the loyalty of consumers with respect to their current suppliers, hence, the higher the barriers to entry for potential competitors (indeed, switching costs are sometimes called "barriers to substitution"). In fact, either new entrants are able to let consumers enjoy the same level of network effects they are accustomed to and to enjoy an operative environment in which they may employ past learning experiences, or they need to compensate consumers for the advantages they loose switching to a new supplier. One possible competitive shortcoming of learning and network effects is that switching costs may become so significant, that users "think twice" before leaving the old network good in favor of a new one, even if the "new entrant" is *per se* technologically superior.[17,18] In fact they are sure to loose (a more or less significant part of) their sunk

---

[12] See, in particular, M. T. CLEMENTS, *Direct and Indirect Network Effects: Are They Equivalent?*, 22 International Journal of Industrial Organization, 633--645 (2004) (but also KATZ & SHAPIRO, *Network Externalities*).

[13] KATZ & SHAPIRO, *Network Externalities*.

[14] As extensively described in the literature concerning Microsoft antitrust cases in the US (and by the DoJ itself on trial), this barrier arises because a user that wants to change his operating system has also to change all his applications (designed to run on that system) and frequently to "port" (convert into a new format) all his files (a long and costly operation, if automatic conversion is not 100% reliable – and usually it is not). Moreover, the user knows that more applications are written for an OS with an high market share and that an OS has an high market share if it has a lot of applications (because users love the possibility to have a lot of applications, sometimes even if they do not really make full use of them all). So, it is evident that a popular OS market power is protected by its own applications and that this process is self reinforcing.

[15] See C. CHRISTIAN VON WEIZACKER, *The Cost of Substitution*, 52 Econometrica, 1085--1116 (1980); PAUL KLEMPERER, *The Competitiveness of Markets with Switching Costs*, 18 The RAND Journal of Economics, 138--150 (1987). But see also JOSEPH FARRELL & CARL SHAPIRO, *Dynamic Competition with Switching Costs*, 19 The RAND Journal of Economics, 123--137 (1988). For a very synthetic survey of this literature, see NET LE, *Microsoft Europe and Switching Costs*, 27 World Competition, 567--594 (2004).

[16] European Commission Notice on the definition of the relevant market for the purposes of Community competition law. OJ C 372 on 9/12/1997, § 42 ("Barriers and costs associated with switching demand to potential substitutes").

[17] In the sense that a single unit of the new good would be preferred to the old one; and even a network of the new goods would be preferred to a similar network composed by the old one. The most famous – but also disputed – example of this "lock in" effect relates to use of the QWERTY keyboard, designed to maximize typing speed under the constraint of avoiding collisions of hammers in old typing machines. Once widespread, this keyboard remained and still is predominant, despite the existence of other models, like the Dvorak keyboard, designed to maximize typing speed without additional obsolete constraints. See PAUL A. DAVID, *Clio and the Economics of QWERTY*, 75 American Economic Review, 332--337 (1985); but see also the critiques of STAN J. LIEBOWITZ & STEPHEN E. MARGOLIS, *The Economics of QWERTY*, *in* History, Theory, and Policy in Essays by Stan J. Liebowitz and Stephen E. Margolis, (Peter Lewin ed., 2002).

learning investments and risk loosing network benefits, if other users do not want to switch (because of different evaluation of the goods or simply because of a coordination failure). This is the phenomenon at the basis of the tipping process, as described by several authors:[19] in case of tipping, there is an excessive "inertia" on the market and users may lack the necessary coordination to migrate to a new superior standard. At the same time, one should also admit that real market failures are not a likely effect of any coordination game: many coordination mechanisms exist, that can help users to migrate to the better standard, at least in the medium run. Moreover, a point worth mentioning is that network effects or switching costs are not (or – at least – not only) strategic barriers generated by the incumbent: loosing the benefit of network effects or incurring switching costs are real costs, frequently impossible to avoid for technological reasons, which the society must bear if a widespread system is substituted by another one.[20]

Summarizing, if we want to isolate cases where network effects produce not only tipping, but also some degree of lock-in and path-dependency, we have to individuate networks characterized by switching costs that are so high as to be prohibitive. And this kind of phenomenon is not likely to arise only from coordination failures, but typically needs to be reinforced by the presence of high learning investments. Several authors actually tend to argue that market inertia is not likely to eliminate competition, but simply to lead to a so-called Schumpeterian (or "serial monopoly") model of competition, where firms tend to compete "for the market" more than "within the market", with an alternation of different temporary monopolists "leapfrogging" each other.

2.3. Models of competition: Schumpeter vs. Arrow

In the literature concerning new technology markets, there are frequent references to two paradigmatic models of competition and innovation. On the one hand, there is the approach I just mentioned, associated with Joseph Schumpeter[21] and related to the idea that a certain degree of market power (or "monopoly") is necessary for innovation and that a kind of "creative destruction" will ensure that real innovations will displace old temporary monopolies and create new (still temporary) ones.[22] Indeed, some empirical evidence of this phenomenon may surely be seen in software markets, when a new "killer application" arises, creating a disruptive innovation.

On the other hand, we have the somehow antithetic approach that competition favors innovation and that monopolies are likely to be more persistent than one may hope and may tend to become relatively inefficient (and yet not enough to be easily surpassed by new entrants, whose brilliant projects may still lack sufficient financial means and marketing tools). This view is frequently associated with Kenneth Arrow. The author was not opposed to intellectual property, but he observed that a firm already in monopoly position – when inventing – faces a higher opportunity cost than a competitive undertaking, in so far as he/she forfeits current monopoly profits.[23] In other words, what is at stake for a competitive firm is the possibility of

---

[18] For instance LE, *Microsoft Europe*, , p. 572 reports that: "Switching costs can create a bias in consumer choice against a new and arguably better product. The choice of OS between Windows and Linux is an example. The study of Gartner Dataquest shows that when users switch from using Microsoft Windows to Linux, they must replace or rewrite many Windows applications. The study shows that, on average, the utility surplus created by the Linux package compared to Microsoft's package constitutes only 20±30 percent of the total switching costs that the consumer must incur when they switch to the Linux package. The higher the number of Windows application programs, the higher the switching costs will be. The study concludes that switching from Windows to Linux is profitable only for users who have a narrow demand for applications, or for those who use the older versions of Windows (such as Windows 95), where the number of supported software programs on this platform is limited."

[19] Pardolesi, R. and Renda, A. – "*How safe is the king's throne? Network externalities on trial*", Chapter 11 of "*Post-Chicago developments in antitrust law*"; Lemley, M.A. and D. McGowan (University of Texas and Berkeley) – "*Legal Implications of Network Economic Effects*", 86 California Law Review, 479, 1998.

[20] See, on this specific point (since some other points of the paper may be slightly biased in favor of the dominant software house Microsoft), GEORGE B. RICHARDSON, *Economic Analysis, Public Policy and the Software Industry*, DRUID Working Paper No. 97-4 (April, 1997).

[21] The main reference is to JOSEPH A. SCHUMPETER, *Capitalism, Socialism, and Democracy*, (Harper and Brothers, New York. 1942), p. 83 and Chapter VII (The Process of Creative Destruction) in general.

[22] *See* JONATHAN B. BAKER, *Beyond Schumpeter vs. Arrow: How Antitrust Fosters Innovation*, Available at SSRN: http://ssrn.com/abstract=962261 (published in 74-3 Antitrust L. J. 575--602 (2007)) (June, 2007), p. 4—5. *See* also ANDREA OTTOLIA & DAN WIELSCH, *Mapping the Information Environment: Legal Aspects of Modularization and Digitalization*, 6 Yale Journal of Law and Technology, 174 (2004), f.n. 156—160 and accompanying text.

[23] KENNETH J. ARROW, *Economic Welfare and the Allocation of Resources for Invention*, *in* The Rate And Direction Of Economic Activities: Economic And Social Factors, 609-626 (Richard Nelson ed., 1962), pp. 175—178 of the reprinted version. See also pp. 170—171 about the so-called "paradox of information," constituting an important justification for the existence of intellectual property rights. *See* also BAKER, *Beyond Schumpeter vs. Arrow*, p. 5—6: and f.n. 8: "Arrow observed that a monopolist bears a cost

conquering a market, while an established monopoly already enjoys significant market power and he may decide to manage the pace of innovation in such a way as to avoid risky threats from potential competitors, but without unnecessarily disrupting solutions which are "selling well". Hence, several authors criticizing the pseudo-Schumpeterian theory of monopoly profits to provide incentives to innovate make reference to Arrow, arguing that it is the pressure coming from competition, which urges entrepreneurs to innovate, while firms holding quasi-monopoly power may prefer to invest in creating barriers to entry, instead of betting on research and development efforts.[24] In software markets, this may be done, for instance, withholding access to interoperability information or using tying strategies to acquire distributional (and other) advantages.[25]

I have no ambition to seek to clarify the pseudo-Schumpeter vs. pseudo-Arrow debate, nor of deepening it. In fact, I sympathize with contributions showing that there is a great deal of complementarity among the aforementioned approaches.[26] The main point that interests me, and that will be seen again in the following papers, is the simple intuition that firms will innovate whenever performing an investment in innovation improves the competitive position of the undertaking more than not innovating (clearly considering also that innovating may increase the total surplus available for consumers, hence also the share one may extract from them). To integrate such a condition, a certain degree of appropriability for innovations is necessary (as shown by Schumpeter, among others), but it is also necessary that not-innovating incorporates a significant risk of being caught-up by competitors (as shown by the literature looking more to Arrow's approach). Hence, intellectual property should exist, to increase rewards from innovation, but it should not offer an excessively broad and/or long protection, in order not to reduce, too much, the threat coming from potential competitors. As I will discuss (in particular in the second paper), an effective combination of incentives to innovate and threats in case of lack of innovation may be created coupling copyright and trade secret, but leaving room for extensive freedom to engage in reverse engineering. Another important point, that I will discuss more later on, concerns the necessity to balance the need for incentives to create in the first place, and the need to allow incremental innovation, without creating excessive transaction costs, coming from a plethora of cross veto powers, related to excessively broad intellectual property rights.

2.4. Incremental and cumulative innovation

Developing software is a complex undertaking, normally faced by teams of skilled developers, frequently exercising a highly creative work; however, it has been observed that the bulk of software development does not entail the kind of "inventive step" needed in order to be granted patent protection.[27] It has also been observed that, in software markets, it is frequently difficult to distinguish "innovators" and "borrowers". In fact, given the existence of network effects and of the other phenomena I briefly discussed, in order to successfully introduce an innovation it is frequently necessary to borrow something from existing projects, in order to achieve, at least, partial compatibility. As Reichman puts it,

> "most of [the] members [of the technical community] do not pertain immutably to either the category of innovators or that of borrowers. […] In this respect, incremental innovators depend on their predecessors in ways that are more reminiscent of artistic creators than of path-breaking inventors."[28]

Indeed, that is even truer for software developers, because software is a typical "cumulative systems technology, which, as its name suggests, is a technology that builds on and interacts with many other features of existing technology to create a new technology."[29] This is why it has been argued that granting excessively

---

when innovating that an innovating competitor does not, as it gives up the opportunity to continue to earn monopoly profits without innovating. In consequence, the incremental gains from innovation to the monopolist may be less than those of a firm in a competitive setting that would expect to earn similar post-innovation profits."

[24] See, in particular, JOSEF DREXL, *IMS Health and Trinko - Antitrust Placebo for Consumers Instead of Sound Economics in Refusal-to-Deal Cases*, 35 International Review of Intellectual Property and Competition Law, 788--808 (2004): "In a situation in which a company holds market power, this company […] will not 'feel the pressure' to innovate. Or the company will be incited to create barriers to entry for potential competitors and forget about the need to convince consumers to remain faithful to its own products by continued introduction of superior technology in the market."

[25] OTTOLIA & WIELSCH, *Legal Aspects of Modularization and Digitalization*, f.n. 156—160 and accompanying text.

[26] *See* BAKER, *Beyond Schumpeter vs. Arrow*, (and also SIDNEY G. WINTER, *The Logic of Appropriability: From Schumpeter to Arrow to Teece*, (September, 2006).

[27] SAMUELSON, et al., *A Manifesto*, p. 2346.

[28] J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, see id., 2432 , p. 2535.

[29] RICHARD R. NELSON, *Intellectual Property Protection for Cumulative Systems Technology*, see id., 2674 , pp. 2675—2676. See also ROBERT P. MERGES & RICHARD R. NELSON, *On the Complex Economics of Patent Scope*, 90 Colum. L. Rev., 839 (1990), p. 881.

broad or formal property rights in such an environment may be problematic, because of the risk of burdening developers with unnecessary transaction costs and a plethora of cross-*veto* powers.[30]

It is starting from this (synthetic) economic and technological background that I want to provide a few more additional introductory concepts, related to the intellectual property protection of copyright (understanding intellectual property in a broad way, encompassing also trade secret protection).

3. A foreword about the legal tools incentivizing innovation in software markets

Coherently with the idea that software is essentially a cumulative systems technology, in the following papers, I will argue that there are sound economic reasons suggesting that copyright should be the unique means of protection of software. In fact, copyright allows for a significant level of protection against pure free-riding, for instance in the form of literal copying and software piracy; at the same time, however, it permits a quite a free circulation of ideas, technical principles and new methods. In this section I summarize some law & economics of the copyright and patent systems, in order to show that the aforementioned approach – that can be inscribed in the context of the so-called "idea/expression dichotomy" and that I will discuss specifically in the first and second paper of the dissertation – is consistent with economic insights (while a higher level of protection could be offered by the patent field, but would be likely to give rise to several problems).

A first *caveat* concerns the fact that economists have sometimes a tendency to describe in very similar terms the structure of incentives coming from the granting of copyright or of a patent (or other intellectual property rights). In fact, intellectual property rights are frequently described as a tool to generate incentives for the creation of intangible goods (i.e. something near to "ideas", with strong non-rival and – what really creates problems – non-excludable characteristics) through the increase of market power (i.e. granting a temporary legal monopoly, that is, excluding power granted by the law). As observed by Reichman,

> "[m]ost economists challenge every grant of intellectual property rights as an unwarranted and potentially harmful derogation from the competitive ethos, and even the patent and copyright models have won only grudging acceptance in this regard. But precisely because hostile economists almost uniformly prefer to let the market solve its own problems, they seldom question whether the standard legislative response might prove technically deficient for other reasons."[31]

Indeed, patents and copyright are frequently described as very similar or as only quantitatively different, in that patents offer a stronger protection than copyright. In the following paragraphs I will try to briefly discuss whether this is really the case. Or, better, whether this is really the whole story.

The "Great Conventions" of Paris and Berne[32] divided intellectual property into two broad categories: industrial property on one side and literary and artistic property on the other side. Therefore, it is clear that this "great divide" was based on the nature of the protected works and of their value for human beings: utilitarian innovations (protected primarily by patents) on one side and aesthetic creations (protected by copyright) on the other.[33] Several scholars addressed the problems coming from the extension of the copyright paradigm – devised to protect aesthetic creations – to utilitarian works, and I will not try to deny that these problems may be real and are frequently significant.[34] However, years of jurisprudential and legislative efforts have shown quite clearly that copyright is flexible enough to protect utilitarian works as well.[35] Moreover, the peculiar form of software legal protection – resulting from the combination of copyright and trade secret – is not simply "weaker" than the one offered by patents; it has its own advantages and peculiarities.

As I will try to show, patents are near to guarantees on the appropriation of the social surplus generated by a new idea,[36] while copyright is nearer to a guarantee on the *possibility* of recouping sunk up front costs, *through*

---

[30] RICHARD R. NELSON, *Intellectual Property Protection for Cumulative Systems Technology*, 94 Columbia Law Review, 2674 (1994), p. 2676.

[31] REICHMAN, *Legal Hybrids*, p. 2505.

[32] Paris Convention for the Protection of Industrial Property (1883); Berne Convention for the Protection of Literary and Artistic Works (1886).

[33] For a deep analysis about this topic, see REICHMAN, *Legal Hybrids*.

[34] See, for instance, GUSTAVO GHIDINI, *Profili evolutivi del diritto industriale. Proprietà intellettuale e concorrenza*, (Giuffrè, Milano. 2001) (or GUSTAVO GHIDINI, *Intellectual Property and Competition Law. The Innovation Nexus*, (Edward Elgar. 2006)); DENNIS S. KARJALA, *Distinguishing Patent and Copyright Subject Matter*, 35 Connecticut Law Review, 439 (2003).

[35] See, in particular, JANE C. GINSBURG, *Four Reasons and a Paradox: The Manifest Superiority of Copyright over Sui Generis Protection of Computer Software*, 94 Columbia Law Review, 2559--2572 (1994).

[36] In a way which is coherent with the so called "prospect theory".

a rule against free riding (on these up front investments).[37] This implies that both legal tools may protect R&D activities, but patents will better protect Research-intensive industries, while copyright will better work in Development-intensive activities, as the incremental-innovation-based software industry. In other words, I suggest that industries in which innovation is very difficult to achieve, but easy to implement, need a patent-like protection; while industries in which copying a new idea is far from sufficient to become a successful new competitor may be protected by copyright.

It is well known that, if something you did is copyrighted, no one else can literally copy it in order to benefit from your effort in creating the (immaterial) good (or the first copy of the material good embedding it) for the first time. But it is also a pillar of the copyright paradigm that your "competitors" can be inspired by your ideas and invest in creating something functionally very similar, based on the same concepts, ideas and algorithms, or even identical in some details. (That is true, in particular, if something of what you did was not your arbitrary choice as artist/creator, but it was mandated by physical/natural laws or even by technical exigencies or other external factors: see the first paper for a fuller discussion of similar issues applied to software.) This copyright-rule, usually, does not confer you a strong market power, because your opponents are normally free to realize close substitutes of what you did, free-riding on your ideas and intuitions, in some ways emulating your genius, even if the copy of the external form of what you did is banned. For instance, if you wrote Harry Potter, they can write stories about young magicians going to a school of magic and maybe having some kind of powerful and mysterious arch-enemy, but they cannot use the name of your character or copy completely arbitrary details, like the fact that your young magician has a bolt-like scar on his forehead.[38] Moreover, if someone, independently from you, writes a story about a young magician called Harry and with a bolt-like scar on his forehead, he may enjoy (at least in several countries) a defense of independent creation.[39,40]

---

[37] For a systematic comparison between patent and copyright paradigms, see KARJALA, *Distinguishing Patent and Copyright*, ; GHIDINI, *Profili evolutivi*, ; MARCO RICOLFI, *La tutela della proprietà intellettuale: fra incentivo all'innovazione e scambio ineguale*, I Rivista di diritto industriale, 511--525 (2002). See also REICHMAN, *Legal Hybrids*. For a specific reference to software, see GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè second ed, Milano. 1997).

[38] Actually, copying one of these details would likely be allowed: it is the copy of a sufficiently broad collection of these details that would likely entail copyright violation.

[39] In the field of (published) literature, one can almost always have access to a previous creation, so credibly claiming an independent creation may be difficult; but software code is usually a carefully protected trade secret, so proving independent creation may be easier, if one can prove that one had no way to access the original code (clearly, it could still have been copied from the binary or decompiled code): in this sense, (published) open source software may be somehow more easily and clearly protected in the pure copyright domain (but receives no trade secret protection!).

[40] Here there is a delicate legal problem, concerning the existence of a "defence of independent creation" in Italy and some other civil law jurisdictions. In particular, in Italy some commentators understand the novelty (see below) requirement as objective. For more details, see *Il diritto d'autore*, in N. ABRIANI-G. COTTINO-M. RICOLFI, *Diritto industriale*, in *Trattato di diritto commerciale* diretto da G. Cottino, Cedam, Padova, 2001, 335-517 and Auteri in P. AUTERI-G. FLORIDIA-V. MANGINI-G. OLIVIERI-M. RICOLFI-P. SPADA, *Diritto industriale. Proprietà intellettuale e concorrenza*, Giappichelli, Torino, 2005. *Contra* GIOVANNI GUGLIELMETTI, *Le topografie dei semiconduttori*, AIDA, 191 (1992). This is my understanding of the Italian situation (at least in theory). In Italy the dominant jurisprudence, usually followed by case law, decomposes the requisite of creativity (to enjoy copyright protection) into the two requisites of originality and novelty, interpreting novelty in an objective sense. (Subjective novelty adopts the point of view of the creator: a creation is new if it is not an imitation of existing creations. This is the typical approach to the requisite of originality in copyright law in common law countries. If we shift to the point of view of the public, we have objective novelty, when no identical expression ever existed before: this approach to novelty is typically followed in patent law.) The objective approach to novelty of Italian author's right law may seem to collide with the approach of this paper. Fortunately, if we examine the general requisite of creativity in a broad enough way, some similarities may not be attributed to a lack of novelty (in the subjective or objective sense), but simply to the fact that a given expression is dictated by the rules or grammar and syntax (especially in the case of the very formalized software languages): in other words, a broad approach to the idea/expression dichotomy may be a substitute to the defence of independent creation.

About Italian law, another problem may concern trade secret law, which recently underwent a major reform. See GUSTAVO GHIDINI & VALERIA FALCE, *Recent developments in Italian regulation of trade and industrial secrets: A patent contradiction of the patent regime?*, paper presented at the 3rd Annual Workshop on the Law and Economics of Intellectual Property and Information Technology, 5-6 July, 2007, Queen Mary, University of London (July, 2007). But see also LUIGI CARLO UBERTAZZI, *Commentario Breve alle Leggi su Proprietà Intellettuale e Concorrenza*, (CEDAM Fourth ed, Padova. 2007). The problem may be that, in some jurisdictions, secrets may be protected as quasi-property rights (and not only against acts of unfair competition) and that it is not clear that reverse engineering is considered a proper way to discover them. Fortunately, Italian law will not be interpreted in an excessively restrictive way, because this would likely be incompatible with Article 39 TRIPs. In any case, at lease in the field of software, there is fortunately a specific exception (due to the reception of the Software Directive) allowing decompilation (which must prevail – being a specific norm – over the general protection granted to secret information).

Notice, however, that in this respect (i.e. the free-riding on ideas), investments in copyrighted items are not much different from investments in the traditional fields of property. The fact of supporting some relevant up-front investment, for instance to buy highly specific machineries (drastically loosing values when they leave their factory and are installed in your firm) in the field of shoe making, does not give you a monopoly on the production of shoes (not even in a small geographical area or for any specific kind of shoes you may have just introduced).[41] So, how can entrepreneurs in the shoe manufacturing receive incentives to invest in the first place? Cleary they know that their investments cannot be appropriated by someone else. Competitors are free to buy the same machines and (usually) to produce almost identical shoes,[42] but they cannot steal the machines of the first producer (there is excludability) and they cannot create confusion for customers regarding the origin of the goods (this because of the law and customs against unfair competition, even without considering trademarks), so that it is possible to "invest" in good reputation. Hence, will ruinous competition compress profit margins so much that – taking into account reasonable expectations – nobody would invest in the first place? Normally not,[43] because new entrants have to finance machineries (and invest in advertising and reputation) in a very similar way to the incumbent (they may have a significant advantage in copying some entrepreneurial intuitions, which are at the disposal of free riders, but they also have severe disadvantages in terms of reputation and advertising, where the incumbent's lead time creates a value that cannot be easily "stolen"). It is the symmetry of entry costs that makes possible the sustainment of these costs in the first place: new entrants will stop entering the industry when they believe they will not be able to recoup their up-front investments.[44] And if they decide to enter because entry is still profitable for them – since they have also several cost/reputation disadvantages with respect to the original developer – the incumbent is also likely to be still making sufficient profits (probably higher than the ones of late comers). Of course, if new entrants are more innovative or efficient than the incumbent, they may displace it, but few commentators would argue that this is an undesirable outcome.

In a world without IP, "late comers" would be allowed to simply duplicate distributed computer programs, so that a market failure would be likely: this is what would happen with plain (and not contrasted) software piracy. Pirates simply pay the marginal cost of reproducing the original immaterial good (e.g. the price of a CD or DVD or needed bandwidth) and just bare negligible sunk costs (typically, some non-R&D-related fixed costs, which are not even completely sunk,[45] as the cost of a CD burner; in some cases, the cost of infringing some technical measure of protection). A situation where piracy was legal would be almost equivalent to allowing competing shoemakers to steal the machinery from the incumbent producer.[46] New entries would occur as long as there was a profit to be made, with respect to the fixed costs necessary for pirates to produce, so that the recoupment of the original innovator's sunk costs could take place only if he was particularly lucky or skilled in exploiting his short lead-time. However, in a software market where piracy was allowed, lead-time would be very short because digital copying is an almost instantaneous process, as the assembly of a production line for CDs or DVDs.

Fortunately, what is strictly needed to avoid major market failures seems to be a narrow legal monopoly of creators to be used against pure free riders (like final consumers and software pirates, so-to-say "copying and

---

[41] Please, notice that the "shoemaker" example is just that – a simplistic example representing any old economy industrial activity – and that I assume that the shoes and the machinery used to produce them are not protected at all by intellectual property rights.

[42] There may be some weak intellectual property protection, but it is usually not very significant and – in any case – I assume it not present for the purposes of this example.

[43] For a more authoritative view, see DREXL, *IMS Health and Trinko*: Drexl criticizes the pseudo-Schumpeterian theory of monopoly profits to provide incentives to innovate: "In a situation in which a company holds market power, this company […] will not 'feel the pressure' to innovate. Or the company will be incited to create barriers to entry for potential competitors and forget about the need to convince consumers to remain faithful to its own products by continued introduction of superior technology in the market. Microsoft's alleged anticompetitive behaviour in bundling its operating system with its own application software provides evidence of such danger."

[44] Here I do not take into account, for simplicity, coordination problems: in practice, I assume that new entrants arrive sequentially/serially and look at profit opportunities, deciding to enter or not depending of the number of people already competing in the sector (and the related profit opportunities). It is actually possible that too many new entrants arrive, especially if we have huge positive expectations (think about the dot-COM bubble at the beginning of the New Economy success) that some of them bankrupt and that the reaching of an equilibrium is actually very "painful", but this situation seems to be quite rare.

[45] A pirate may stop his production, sell the machinery and recover part of these fixed costs: this is not possible for true sunk costs, which are irredeemable.

[46] Actually there is a difference between physical machines and immaterial goods: in the case of immaterial production factors, the absence of rivalry would reduce the likelihood of a market failure, since the original producer would not be deprived of his production tools.

pasting" significant portions of code). It is less clear whether any kind of protection is needed against other developers realizing from scratch very similar pieces of software. Indeed, some authors think that such a broader protection is not necessary: "the policy justification for affording software an increased level of protection compared to traditional technologies is piracy prevention[.] […][An] increased protection should be no greater than necessary to satisfy [this] policy goal."[47] In fact, the simple fact that software piracy is illegal seems to sufficiently increase the costs for pirates, making it possible for original developers to survive, even if some piracy still exists.[48] Even if I am not aware of any evidence to economically quantify the risks incurred by a software pirate, it is reasonable to guess that these costs are fairly low, and surely much lower than the cost of reverse engineering. This is an anecdotic, yet not easily dismissible, evidence that, if software pirates are not able to disrupt software innovation, it is highly unlikely that any kind of software reverse engineering could have this effect (but see the second paper of this dissertation for a fuller treatment of this point).

Rephrasing what I just discussed, the kind of protection offered by copyright comes (at first approximation) from excludability on sunk (development) costs, without granting broad economic monopolies to first comers (and leaving some possibility of free riding on research costs and brilliant ideas in general).[49] Similarly, the first author of a mystery or science-fiction novel does not acquire a monopoly on the genre, but only on his very book, which may have thousands of substitutes, that are more or less close depending on the skills of the authors and the taste of the readers. Indeed, as in the case of software, writing a competing novel of the same genre is not significantly more or less costly than writing the first one, even though subsequent authors may derive a lot in inspiration from the first one. The difference between this model and that of patents is very significant: in the latter, in fact, the state is granting a monopoly on a specific technical solution, which will be reserved to the first inventor, no matter whether late comers are able to independently develop it. If I develop an innovative chemical compound to make concrete hard in two minutes instead of some hours, late comers may not use the same compound without a license from me, no matter if they discover it following a different path of research and discovering an alternative one may be more costly (or the compound itself more costly or less effective). Hence, my discovery (and the relative granting of a patent) may actually and quite directly make third party, which were studying solutions to the same problem, worse off. In principle, if a unique technical solution exists, a patent may offer a full economic monopoly on that solution (for a limited but very significant period, e.g. 20 years). Such an outcome may not be especially problematic in some fields, especially in the ones in which one product may be covered by one or few patents. However, the same cannot be said for complex system technologies.

Nowadays, the digital revolution is making piracy much easier and efficient than in the past and this is posing a significant threat to the day-by-day working of the content industry (including – at least in part – the production of software). However, I do not think that the possibility of digital reproduction is changing the core of the copyright system, at least not as far as the relationships between competing producers of intellectual goods are concerned. What is really new (but mainly in a quantitative sense, if we compare with the case of reprography/photocopies) is that not only other "editors", but also end users now become potential competitors of the original producer. This is a relevant difference when we deal with the enforcement of intellectual property rights and surely also in designing market strategies and norms dealing with technological measures of protection and so on. However, if we deal with other people wanting to access to the interfaces of a given piece of software, the traditional principles of copyright law (as the idea/expression dichotomy that I will discuss more in the first paper) remain relevant and capable of avoiding market failures. In fact, if software is protected by copyright, in the "normal case" of producers of software

---

[47] DENNIS S. KARJALA, *Copyright, Computer Software, and the New Protectionism*, 28 Jurimetrics Journal, 33 (1987), p. 36.

[48] According to the Fifth Annual BSA and IDC Global Software Piracy Study, the worldwide weighted average of piracy rates is 59.9%. Rates vary widely across countries, with the United States at 20%, Japan at 23%, Belgium at 25%, Italy at 49% and China at 82% (with some former Soviet countries bordering 100%). See NationMaster.com, page available at http://www.nationmaster.com/graph/cri_sof_pir_rat-crime-software-piracy-rate; last visited September 15, 2008.

[49] See, for instance, GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè first ed, Milano. 1996), pp. 277—278. The author argues that copyright (*rectius*, author's rights) aims at preventing forms of complete free-riding, which are susceptible of also eliminating forms of competition based on the non-inventive form of innovation, which is characteristic of software: "il diritto d'autore sul software punta ad impedire le forme di imitazione idonee a compromettere la concorrenza basata sull'innovazione non inventiva […]. Le condotte che recano un effettivo pregiudizio […] sono quelle che, attraverso la copiatura totale o sostituzioni che possono essere fatte senza costi sostanziali, consentono di trarre un immediato vantaggio dalla maggior vulnerabilità all'imitazione dei programmi rispetto ad altre tecnologie assistite unicamente dal sistema brevettuale, impedendo agli innovatori di ottenere la remunerazione dei loro sforzi."

satisfying similar needs with respect to the ones met by the incumbent's piece of software (but without major interoperability requirements), late comers will have to invest, up-front, an amount of resources similar to that invested by the first developer in order to produce his software. As for the production of shoes, advantages related to an already clearly marked path of innovation exist, but they should be compared with the disadvantages related to the lead-time of the incumbent. In a way, we could say that there would be a partial free riding on research costs, almost no free riding on development costs related to actually writing down the software, and a more or less relevant advantage of the incumbent because of lead time (and hence reputation, learning effects, network effects, switching costs for users and so on). (I am going to come back to these ideas in § *3.3. Choosing between copyright-like and patent-like protection*.)

3.1. Putting interoperability and reverse engineering into the picture

As I will show in the second paper of this dissertation, in general it is not possible to determine theoretically if the net investment required to late comers to compete effectively in software markets is higher or lower than the sunk cost paid by the incumbent. In fact, network effects, switching costs and other consequences of lead-time tend to favor the incumbent, while late comers may take advantage of some free riding on the ideas and methods originally developed by the first comer. What it is possible to say is that in these markets (as in the market for microchips, which is regulated by very similar intellectual property rights) there seems to be a healthy degree of innovation (it is difficult to say "enough innovation," because we do not have any idea about the optimal pace of innovation!).

A situation deserving special attention is the one in which late comers need to have a certain degree of interoperability with the incumbent's software (or some complementary products originally designed to work with it, or the output files generated by it/them): this regularly happens in the markets for software platforms. In these markets, the first comers become "protected" by the existence of network effects, so that late comers need either to produce such a clearly superior (or cheaper)[50] product to induce customers to forgo network effects (al least temporarily and hoping for a coordination on the new product) or try to enjoy – at least partially – the same network effects. (Similar arguments apply to learning costs sustained by users.) In this case, new entrants do not only need to sustain similar costs to implement a new "original" software, but they also need to gain access to interoperability information. If this is done through reverse engineering, the cost of software decompilation should be added to the sunk costs of late comers, so that they are actually likely to have higher development costs than the incumbent (despite the fact that they may free ride on some of his original research investments).[51] I will not discuss these issues any further here, since I devoted time to them the second paper of this dissertation.

3.2. Patent (or patent-like) protection

When one is granted a patent, one's monopoly is much "deeper" (or "broader", if you like) than when one simply enjoys copyright. In a few words, with a patent one receives an exclusive right to all technical applications of an idea that one has been able to describe in the patent application (as long as this idea is not the normal technical solution that a person skilled in that field would have applied to deal with the issues the inventor was confronted with).[52] Therefore, the economic effect of owning a patent is very different to that of enjoying a copyright, at least in several cases where the patent is actually conferring a significant degree of market power. Of course, market power cannot be evaluated in abstract terms, because it crucially depends on technical and economic elements. (How close are the substitutes of the good that may be produced without infringing the patent? How elastic is the demand for the good at hand?) In any case, it is quite likely that competing with the owner of a patent without infringing it will be relatively costly, so that not only the patent owner is able to recoup his sunk cost (if there is a significant demand for what he invented), but he is

[50] But since marginal costs are almost *nihil*, incumbent (with deep pockets) may always credibly threat to fight a new entrant (with limited financial resources).

[51] MICHAEL A. JACOBS, *Copyright and Compatibility*, 30 Jurimetrics J., 91 (1989), p. 102 refers estimates indicating that the simple translation of a complex program from a given programming language to another one – an operation which is conceptually much simpler than a complete decompilation from the object code – may cost more then half the whole original development cost (also quoted in GUGLIELMETTI, *L'invenzione di software (1st ed.)*, f.n. 92, p. 279).

[52] Clearly, the fact that patents are "granted" after an examination introduces some complexities in the description, because the fact of receiving a patent or not and the actual level of market power granted depends significantly on the skills and competencies of the patent office (and since having hundreds of a country's best engineers working in a patent office would not be feasible, it is evident that the level of these competencies is not always likely to be as high as it should be in order to avoid the granting of trivial patents).

also likely to appropriate a significant part of the social added value of the innovation. In other words – following the traditional story told by any law & economics book when talking about intellectual property – if a benevolent social planner would have been willing to pay X to let society enjoy the innovation during the period protected by the patent, the patent holder is likely to receive a rough approximation of X (minus the deadweight losses created by monopoly), minus something depending on the number of competitive technologies which are "inspired" – but not covered – by the patent (and are realized by other people), minus – of course – the value that will be created by the innovation from the 21st year of its creation onward (notice, however, that this amount need to be actualized to the moment of devising the innovation, hence it's likely to be a relatively small fraction of the total X). To exemplify, think about a patent on a new product; the patent holder will be able to appropriate a significant part of the willingness to pay of the society for this product during the 20 years (or so) of the patent's duration. However, this amount will be smaller if the existence of this new product pushes several other inventors to work around the patent and produce new (non-infringing) substitutes. Moreover, transaction costs and other inefficiencies generated by the fact that the technology is proprietary, instead of being available as a public good, will further reduce social welfare.

Clearly, a permanent monopoly (with perfect discrimination, actually) on new ideas would provide the perfect incentive to create immaterial goods: this incentive is "perfect" (in a static world) in the sense that it is never insufficient. To be sure, however, there may be over-incentive (with respect to what is needed to trigger innovation) and – in the real world – there are huge costs, both in static terms (deadweight losses, if price discrimination is not perfect) and in dynamic terms. Dynamic costs, in particular, concern obstacles to subsequent innovation, even though this may be partially solved by rules on derivative innovations.[53]

Software is, essentially, a combination of algorithms, methods and mathematical formulas. Indeed, the dangers intrinsically present in patenting an object as software have been recognized by several legislators. For instance, Article 52 of the European Patent Convention excludes from patentable subject matters not only "discoveries, scientific theories and mathematical methods," but also "schemes, rules and methods for performing mental acts, playing games or doing business," and specifically "programs for computers". However, Paragraph 3 of the same article specifies that these subject-matter or activities are excluded from patentability "only to the extent to which a European patent application or European patent relates to such subject-matter or activities as such." In the dissertation, I will briefly come back to the interpretation of this provision. For the moment, I just want to point out the fact that, as obscure as this wording may look (and, indeed, is), it had the effect of allowing the granting of thousands of software patents by the EPO. For instance, according to some estimates,[54] "over 11,000 software-invention patents [have been] granted by the EPO as of 1995."[55] At the same time,

> "however, there still remains significant legal uncertainty as to the enforceability of these patents because an EPO patent can still be invalidated by an individual country within the European Union. As a result, very few software patents have been litigated within Europe because of the uncertainty and fear of being invalidated by a national court."[56]

This uncertainty is reflected in the actual strategic working of software patents, which has been effectively described in the following way: "The software patent game is like the Cold War: The only thing that protects you is the concept of mutually assured destruction."[57] In other words, even if nobody may be sure about the enforceability of a specific software patent, the utility of software patents in general mainly consists in their retaliatory power and in the possibility – given a significant pool of rights – to cross license them with other firms, in order to prevent legal problems. A similar description of the world of software patents is shared also in the business world. The following story, told by the lawyer and Silicon Valley entrepreneur Gary L. Reback, may be instructive:

---

[53] See GHIDINI, *Profili evolutivi*, (or GHIDINI, *IP and Competition Law*,); PAOLA A. E. FRASSI, *Innovazione derivata, brevetto dipendente e licenza obbligatoria*, I Rivista di Diritto Industriale, 212--226 (2006).

[54] Derived from the European Patent Office 1994 Annual Report, Patenting Computer Software (1995).

[55] MICHAEL CHAPIN, *Sharing the Interoperability Ball on the Software Patent Playground*, 14 Boston University Journal of Science and Technology Law, 220 (2008), f.n. 55.

[56] Id., p. 230.

[57] Anti-patent campaigner Florian Mueller, commenting the creation of the "patent commons" project, launched by Open Source Development Labs (OSDL) "to collate details of all software patents pledged or donated to the open source community". *See* OUT-LAW News, *Patent repository to aid open source development*, 11/08/2005 (http://www.out-law.com/page-5999 last visited: July 7, 2008).

> "My own introduction to the realities of the patent system came in the 1980s, when my client, Sun Microsystems—then a small company—was accused by IBM of patent infringement. […] After IBM's presentation, our turn came. As the Big Blue crew looked on (without a flicker of emotion), my colleagues—all of whom had both engineering and law degrees—took to the whiteboard with markers, methodically illustrating, dissecting, and demolishing IBM's claims. […] Only one of the seven IBM patents would be deemed valid by a court, and no rational court would find that Sun's technology infringed even that one.
> An awkward silence ensued. The blue suits did not even confer among themselves. They just sat there, stonelike. Finally, the chief suit responded. 'OK,' he said, 'maybe you don't infringe these seven patents. But we have 10,000 U.S. patents. Do you really want us to go back to Armonk [IBM headquarters in New York] and find seven patents you do infringe? Or do you want to make this easy and just pay us $20 million?'
> After a modest bit of negotiation, Sun cut IBM a check, and the blue suits went to the next company on their hit list."[58]

Today Sun Microsystems is a big firm and its relationship with IBM completely changed. Instead of signing a check, today a cross license agreement could likely be signed. Moreover, IBM's patent policy has completely changed in the meantime, and the firm is today part of several initiatives to protect small open source firms from similar behaviors. However, other firms act in a similar way and the reality of software patents has not changed that much. Indeed, also some changes in the opposite direction, with respect to the one experimented by IBM, took place and some former opponents of software patents may, today, be less strict in their resistance to this phenomenon. For instance, also some of the symbols of today's commercial software houses share (or shared) a somber view of patents as a tool to foster software innovation. Indeed, in 1991, Mr. Gates wrote:

> "If people had understood how patents would be granted when most of today's ideas were invented and had taken out patents, the industry would be at a complete stand-still today. The solution […] is patent exchanges […] and patenting as much as we can. […] A future start-up with no patents of its own will be forced to pay whatever price the giants choose to impose. That price might be high: Established companies have an interest in excluding future competitors."[59]

Mr. Gate's statement underlines a crucial point, highlighted also in the literature about software patents (and on which I will come back). In fact, "[r]ecognizing the public's interest in software interoperability, the private sector has implemented partial solutions such as standard setting bodies, cross license agreements and open source."[60] Indeed, also the open source model of software development can be seen as (among several other things) a way to "contract around" the typical market failures of the copyright and (partially) patent systems in the field of software. But, while copyright seems to be flexible enough to allow this contracting around its possible inefficiencies, patents may create much more problems. (See also the second paper, § *8.1. The Limits of Technology Copyright and Its Natural Antibodies*.)

3.3. Choosing between copyright-like and patent-like protection

To generate incentives, we can work in a continuum as that which follows: patents – copyright – trade secret – unfair competition – social norms against slavish copying (and praising innovative behavior). Moreover, we should take into account how the previous legal/social norms interact with the technological and economic aspects of lead-time. Notice also that the forms of protection to the "right" of trade secret are normally cumulative. What is more, in the field of software, it is commonplace to cumulate copyright and trade secret protection.[61] In several jurisdictions and for several subject matters it is actually possible to cumulate also patents and copyright (but patents require a certain degree of disclosure and hence may reduce trade secret protection).

As I will discuss more in the papers, granting patent protection to software innovations may entail significant social costs, which do not seem to be compensated by the potential increase in incentives to innovate. On the one hand, there are fields of *research* where the likelihood of obtaining some marketable

---

[58] Gary L. Reback, Patently Absurd, 06/24/2002(http://www.forbes.com/asap/2002/0624/044.html last visited: July 7, 2008).

[59] Quotation referred by FRED WARSHOFSKY, *The Patent Wars* 170-171 (New York, Wiley. 1994).

[60] CHAPIN, *Sharing the Interoperability Ball*, p. 234.

[61] Only the object code is published, but it is a general principle of copyright law that unpublished works are copyright protected as well: for this reason, it is possible to couple copyright protection (on object code and source code) and trade secret (on source code) to protect software. See the first paper at § *2.1. Access to Interoperability Information* for more explanations about source code and object code.

results for one's research are quite low (i.e. it is possible to do research for years without obtaining any working product) and where – at the same time – late comers are very likely to be able to copy the first comer's main ideas at a very low cost (for instance, this is the case in the chemical and pharmaceutical sector, where it is frequently easy to reproduce a given compound or drug, once its main active principles have been discovered, tested and described). On the other hand, when dealing with typical computer programs, we do not have a huge level of uncertainty about the fact that what we are doing will work in a more or less decent way, even if there may be an higher level of uncertainty about the response of the public. However, unless literal copying is allowed, rewriting a piece of software from scratch, even if a functionally equivalent product has already been realized, is likely to entail *development* costs being almost as high as the one to develop the first piece of software arrived on the marker. This is one of the reasons why my suggestion is that copyright protection is a better tool to protect innovations for which the "development" part of "research and development" costs is dominant; while, in the fields that are better protected by patents, the "research" part is supposed to be dominant.

Theoretically, there may be cases in which a very high level of uncertainty may be associated to completely new software technologies: this is a field of technology as any other[62] and – despite the existence of several reasons for avoiding a proliferation of software patents – I am not persuaded that no patents at all should be granted in this specific field. This is not to say that I am in favor of generalized software patents: there are several reasons, essentially based on transaction costs (in a broad sense), not to like software patents at all. Nevertheless, I think that a legislator could legitimately decide to grant some patents also in the field of software, as long as he is aware that some general problems of the patent system are especially significant in this domain. First or all, software innovation is typically incremental and cumulative, so that obstacles to subsequent innovation created by patents may be severe. In fact, software systems are made of several components, each of which becomes a kind of input for the others, so that conferring a monopoly on a given good may actually hinder a lot of follow-on developments, in particular because of the absence of an independent creation defense (available in the field of copyright). Moreover, the average duration of patents (20 years) is disproportionate with respect to the pace of innovation in this market (nearer to 3-5 years).[63] Additionally, several commentators suggest that there is a significant lack of skills, in patent offices, as far as software technology is concerned (even though I concede that, if this was the only problem, it should not be solved by excluding software from patenting, but hiring software engineers in patent offices).

Overall, what I am arguing is just that – in case of really new technologies implemented via software – the natural kind of protection should be a patent (an incentive to invest in risky projects with results which are significantly improving the state of the art) and not a broadening of the reach of standard software copyright (in the direction of protecting ideas merged with their expression), because this alternative way would create huge costs in terms of unnecessary protection of obvious incremental improvements created by software, or – what is worse – would afford long lasting market power to products without any value, apart from those derived from having been the first in their field (with the likely theoretical effect of an over-incentive to rent seeking and with the even more likely practical effect of strengthening the position of actors which are already leaders in the software market and of distorting their competitive strategy in the direction of maintaining and strengthening the protection offered by network effects and lock-in of users).[64] Moreover, and recalling the reasoning concerning cumulative system technologies, unsuitable property rights may create a "tragedy of the anti-commons,"[65] which is much more likely in a patent-based system and in sectors where incremental innovation and complementary systems are common. Another reason to prefer copyright to patents in the field of software. My general conclusions concerning IP protection of software are largely consistent with the ones proposed, for instance, by Dam:[66]

---

[62] *See* SAMUELSON, et al., *A Manifesto*.

[63] *See* Id., Also Amazon's Bezos – despite the fact that his firms owns the world-famous one-click e-commerce patent (or, actually, because of that) – argued in an open letter that similar business methods and/or software related patents should not be granted 17 years of protection, but only 3 to 5 years. *See* Russ Mitchell and Anne Speedie, *Amazon's Bezos calls for radical change in patent laws*, The Register, March 10, 2000 (available at http://www.theregister.co.uk/2000/03/10/amazons_bezos_calls_for_radical/; last visited May 10, 2008.)

[64] See the example of Microsoft against Java.

[65] *See* MICHAEL A. HELLER, *The Tragedy of the Anticommons: Property in the Transition from Marx to Markets*, 111 Harvard Law Review, 621--687 (1998).

[66] For a fuller treatment of these issues, see K. W. DAM, *Some Economic Considerations in the Intellectual Property Protection of Software*, 24 The Journal of Legal Studies, 321--377 (1995).

> "First, existing copyright and patent law provides a sound basis for an economically efficient system of protection. Second, the copyright law as currently applied deals adequately with the appropriability problem, without creating significant monopoly or rent-seeking problems. Third, copyright law provides a sound basis for preserving a balance between innovation today and innovation tomorrow".[67]

I am convinced that the patent system, applied to software, would spur almost no additional innovation and would risk creating – because of excessive transaction costs both for original and follow-on innovators – significant side effects. In any case, even scholars thinking that software patents may be economically sound, in principle, frequently admit, at least, that

> "as actually administered […], the system may not adequately balance innovation today versus innovation tomorrow because it is susceptible (until important administrative changes are implemented) to generating too many invalid patents".[68]

Having said that, copyright is very likely to be a preferable tool of protection for software innovation than patents (at least, as they are currently administered), one must also admit that some characteristics of copyright law – like the extremely long duration of protection, devised in such a way as to sustain authors experiencing late success and/or old artists and their heirs – clearly do not make much sense in the field of useful creations, like software. About duration, however, almost the same could be said concerning patents and their 20 years long term of protection, if applied to software. Indeed, the long duration of copyright is a paradox that would deserve a broader critique.[69] At least, in the case of copyright, we face a *de facto* perpetual protection with relatively low intensity, while, in the case of patents, an excessively long protection (with respect to the pace of innovation in software markets) would be coupled with stronger exclusive rights. In other words, in the case of copyright, the disproportionately long term of protection is compensated (in part) by the flexibility of the tool of protection, something that cannot be said about patents. Despite the fact that a clear-cut overall conclusion may likely not be reached, some elements may be inferred. For instance, borrowing a few conclusions from Ramello,[70] one may argue that

> "appropriation by means of intellectual property, similarly to what happens with trade tariffs or taxes, acts as a sort of tax on the production of knowledge: a certain level of appropriation has positive effects on productivity up to a given threshold, after which the effects will be negative.
> At this juncture, the next logical step for economists would be to find the optimal level of appropriation […]. Unfortunately, this is easy to do in theory and almost impossible in practice […]. Considering the high variability of creative and inventive environments, the differences in knowledge produced by different intellectual property rights and the number of all variables defining [the level of appropriation], the search for this figure could be fruitless.
> Nonetheless, from a qualitative perspective the previous result has an important consequence in terms of normative implications. Indeed, at least one important policy prescription can be inferred: the total appropriation of knowledge will never lead to an efficient outcome. Accordingly, the general rule that emerges is that weak intellectual property rights will likely have a more efficient outcome than strong ones."[71]

In particular, in the first two papers of this dissertation, I will try to show why and how relatively "weak" intellectual property rights in the field of software may effectively foster creation and a dynamic, competitive

---

[67] Id., 376—377. But see the first paper for more details (in particular § *4.5.2. Decomposing the Access Phase and the Re-Implementation Phase* and § *7. Vertical and Horizontal Access; Transformative and Substitutive Uses*) for a specific point – relevant for the paper at hand – in which I express some doubts as to Dam's conclusions concerning the different treatment that should be reserved to horizontal and vertical access to computer programs produced by third parties. (In fact, Dam argues that "the distinctions between attachment [vertical access] and replacement [horizontal access] and between transformative and substitutive uses […] do not always emerge sufficiently clearly in the legal literature on software." I will argue that – despite the economic significant of these distinctions – the protection offered by copyright law – possibly in combination with a limited and careful use of patent law – is nevertheless sufficient to stimulate software innovation.)

[68] Id., 377.

[69] In fact, there are just weak justifications – apart from capture of legislators from powerful lobbies – also to protect aesthetic creations for 50 or 70 years after the death of the author. (Indeed, the US Copyright Term Extension Act of 1998 (establishing a term of protection of 95 years for corporate works) gained the nickname of Mickey Mouse Protection Act.) In any case, the duration of copyright protection could hardly have been justified in front of legislators, if it concerned only software: for instance, the US Semiconductor Chip Protection Act of 1984 granted 10 years of exclusivity in the field of mask works – i.e. layouts or topographies – of integrated circuits.

[70] *See* GIOVANNI B. RAMELLO, *Access to vs. Exclusion from Knowledge: Intellectual Property, Efficiency and Social Justice*, POLIS Working Papers, n. 100 (Department of Public Policy and Public Choice - POLIS) (November, 2007) and section 3 in particular.

[71] Id., 18.

environment. However, a risk of choosing a weak intellectual property protection for software, as the one offered by copyright, is that undertakings will increasingly recur to trade secret and – what is more worrying – that some courts and legal systems may be temped to even expand the degree of protection offered to software by trade secret and/or unfair competition regimes.[72]

3.4. Trade secret

Trade secret, as long as it lasts, is surely a socially suboptimal tool of protection. The protection it guarantees is not related to the social value created by the protected knowledge, but just to the degree of excludability of the protected information. Moreover, in case one wants to trade it, Arrow's paradox of information if fully at work, so that any bargaining risks being blocked by a combination of non-credible claims and risks of loosing control over valuable information.[73] All that unavoidably leads to a very limited circulation of protected information and, possibly, to the creation of significant transaction costs. However, several scholars (maybe unexpectedly) concluded that "the law of trade secrets may have surprising efficiency properties".[74] Such a conclusion may be better understood making reference to the writings of Reichman.[75] Following, and sometime criticizing, Reichman's works, several authors take the view that in technological field characterized by incremental innovation, "systems technology, i.e. 'architectural' or meta-technology that determines what can connect to the system and what can operate within it (in this case platforms in modularized systems), is to be protected only by liability rules".[76] These liability rules (about which I will come back in a few lines) need not take the form of generalized compulsory licensing. In Reichman, some quasi-liability rules took the form of "artificial lead-time"; in other writings, an ad hoc access regime, existing – so to speak – in the shadow of antitrust intervention has been preferred.[77] This is typically justified making reference to the combination of "four characteristics of the markets for this technology: network externalities, interconnectivity, rapid innovation, and excludability."[78]

As observed by Reichman,[79] the combination of trade secret and of the possibility of performing reverse engineering works like an "implicit liability rule": to access a given innovation, one has to pay a given amount of resource to discover it by proper means (in order not to violate trade secret law). But there is more: the original innovator is quite sure to make an interesting proposal to potential followers, if he offers them – under a non-disclosure agreement[80] – access to the secret information for a price lower or equal to the expected cost of reverse engineering[81] (in fact, the quality of the information disclosed by the creator is typically superior, with respect to what may be attained through self-help in the form of reverse engineering; at least, this is surely the case in the field of software, as I will show in the second paper). The advantage of this "implicit liability rule" over other forms of protection lies primarily in its capacity to overcome some of the potential failures of a system like the patent one, which are frequently related to transaction costs and/or strategic behaviour. Obviously, its disadvantage – as for any liability rule – derives from the fact that the

---

[72] Indeed, some kind of compensation for copyright's weaknesses may also have been at the roots of some distortions of the patent system in fields such as software. *See* REICHMAN, *Legal Hybrids*, , pp. 2487—2488: "Copyright protection of computer programs in the United States has thus relegated the commercially valuable know-how embodied in publicly distributed programs to a legal limbo. This, in turn, gives new life to the drive for patent protection of computer programs, with its ensuing tendencies to lower the nonobviousness standard and to distort other traditional patent principles. […] Copyright protection of industrial literature thus represents a shortsighted solution that seems likely to trigger the same cycles of over- and underprotection that characterize the history of industrial art." I disagree, however, about the fact that copyright as the main tool of protection for software should be abandoned for similar reasons. Instead, one should try to convince legislators and courts that also a weak form of protection may, indeed, be a good protection.

[73] ARROW, *in* Allocation of Resources for Invention, pp. 170—171 (of the reprinted version).

[74] DAVID D. FRIEDMAN, et al., *Some Economics of Trade Secret Law*, 5 The Journal of Economic Perspectives, 61--72 (1991), p. 71.

[75] REICHMAN, *Legal Hybrids*,

[76] OTTOLIA & WIELSCH, *Legal Aspects of Modularization and Digitalization*, , pp. 228—235 (in particular, text accompanying footnotes 210 and 211).

[77] Id., see in particular pp. 228—235.

[78] Id., pp. 228—235 (in particular, text accompanying footnotes 210 and 211).

[79] REICHMAN, *Legal Hybrids*, p. 2521.

[80] Notice that – while non-disclosure agreements may be appropriate in the context of disclosure of a trade secret in exchange for a consideration – not any kind of clauses may be inserted in these agreements. In particular, some especially harsh clauses preventing the licensee from competing against the licensor in some way or from using unprotected ideas could be considered copyright misuse (if the trade secret is also copyright protected) or otherwise unenforceable or unconscionable. See also Id., , p. 2523-2524. About copyright misuse, see for instance, Lasercomb v. Reynolds, 911 F.2d 970 (4th Cir. 1990), arguing that restrictions contained in licenses cannot operate in a "manner adverse to the public policy embodied in copyright law" (§ 978).

[81] *See* Id., p. 2523.

established price may not be perfect (here, the price is determined by the technological difficulty of performing reverse engineering, not by any planner, but this does not modify the fact that the market cannot fully exploit its informational functions). On the one hand, this implicit price may be too low and incapable of stimulating innovation (but I will show that this case is quite unlikely). On the other hand, it may be that the cost of reverse engineering is too high and no innovation ends up being shared, even in cases in which it would be socially desirable to do so. Notice, however, that in these cases licensing could still take place, so that market failures would not – in any case – be more frequent than under a property right paradigm (even though the information paradox of Arrow would suggest that the creation of clearer property rights could reduce transaction costs, if the typical outcome falls in this category). That having been said, between these two polar cases there is a continuum of possibilities, in which the cost of self-help through reverse engineering lies in between the minimum price that the owner of information would consider as acceptable and the maximum amount that the potential buyer would be willing to pay. In these cases, the quasi-liability rule may make possible several transactions, which could have been prevented by the combinations of overbroad property rights and significant transaction costs.

Notice that, for the moment, I just provided a sketched introduction to this topic. I will diffusely come back to these issues in the second paper of the dissertation. For now, let me just also recall that it has been observed,[82] and it is acknowledge by several commentators,[83] that an excessively easy access to existing platform technology would increase competition within the existent dominant platform, but could actually deter competition between platforms. This is a legitimate worry and should not be underestimated. Throughout this dissertation, I will constantly try to propose interpretations of the law and policy recommendations, which are aimed at making possible competition within existing software platforms, when this is the only way to allow the survival of some competition. At the same time, I will make any effort to avoid recommendations making access to existing (*de facto*) standards so cheap as to deter competition between (*de facto*) standards, when such a competition is a technically and an economically feasible alternative.

[82] P. J. WEISER, *The Internet, Innovation, and Intellectual Property Policy*, 103 see id., 534--613 (2003).

[83] *See*, for instance, OTTOLIA & WIELSCH, *Legal Aspects of Modularization and Digitalization.*

Bibliography

KENNETH J. ARROW, *Economic Welfare and the Allocation of Resources for Invention*, *in* The Rate And Direction Of Economic Activities: Economic And Social Factors, 609-626 (Richard Nelson ed., 1962)
JONATHAN B. BAKER, *Beyond Schumpeter vs. Arrow: How Antitrust Fosters Innovation*, Available at SSRN: http://ssrn.com/abstract=962261 (published in 74-3 Antitrust L. J. 575--602 (2007)) (June, 2007)
DENNIS W. CARLTON & JEFFREY M. PERLOFF, *Modern Industrial Organization*, (Denise Clinton ed., Addison-Wesley Third ed, Reading, Massachusetts. 2000)
MICHAEL CHAPIN, *Sharing the Interoperability Ball on the Software Patent Playground*, 14 Boston University Journal of Science and Technology Law, 220 (2008)
M. T. CLEMENTS, *Direct and Indirect Network Effects: Are They Equivalent?*, 22 International Journal of Industrial Organization, 633--645 (2004)
K. W. DAM, *Some Economic Considerations in the Intellectual Property Protection of Software*, 24 The Journal of Legal Studies, 321--377 (1995)
GIUSEPPE DARI-MATTIACCI & FRANCESCO PARISI, *Substituting Complements*, 2 Journal of Competition Law and Economics, 333--347 (2006)
PAUL A. DAVID, *Clio and the Economics of QWERTY*, 75 American Economic Review, 332--337 (1985)
JOSEF DREXL, *IMS Health and Trinko - Antitrust Placebo for Consumers Instead of Sound Economics in Refusal-to-Deal Cases*, 35 International Review of Intellectual Property and Competition Law, 788--808 (2004)
JOSEPH FARRELL & CARL SHAPIRO, *Dynamic Competition with Switching Costs*, 19 The RAND Journal of Economics, 123--137 (1988)
JOSEPH FARRELL & PHILIP J. WEISER, *Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age*, 17 Harvard Journal of Law & Technology, 85 (2003)
PAOLA A. E. FRASSI, *Innovazione derivata, brevetto dipendente e licenza obbligatoria*, I Rivista di Diritto Industriale, 212--226 (2006)
DAVID D. FRIEDMAN, et al., *Some Economics of Trade Secret Law*, 5 The Journal of Economic Perspectives, 61--72 (1991)
GUSTAVO GHIDINI, *Profili evolutivi del diritto industriale. Proprietà intellettuale e concorrenza*, (Giuffrè, Milano. 2001)
GUSTAVO GHIDINI, *Intellectual Property and Competition Law. The Innovation Nexus*, (Edward Elgar. 2006)
GUSTAVO GHIDINI & VALERIA FALCE, *Recent developments in Italian regulation of trade and industrial secrets: A patent contradiction of the patent regime?*, paper presented at the 3rd Annual Workshop on the Law and Economics of Intellectual Property and Information Technology, 5-6 July, 2007, Queen Mary, University of London (July, 2007)
JANE C. GINSBURG, *Four Reasons and a Paradox: The Manifest Superiority of Copyright over Sui Generis Protection of Computer Software*, 94 Columbia Law Review, 2559--2572 (1994)
GIOVANNI GUGLIELMETTI, *Le topografie dei semiconduttori*, AIDA, 191 (1992)
GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè first ed, Milano. 1996)
GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè second ed, Milano. 1997)
MICHAEL A. HELLER, *The Tragedy of the Anticommons: Property in the Transition from Marx to Markets*, 111 Harvard Law Review, 621--687 (1998)
MICHAEL A. JACOBS, *Copyright and Compatibility*, 30 Jurimetrics J., 91 (1989)
DENNIS S. KARJALA, *Copyright, Computer Software, and the New Protectionism*, 28 Jurimetrics Journal, 33 (1987)
DENNIS S. KARJALA, *Distinguishing Patent and Copyright Subject Matter*, 35 Connecticut Law Review, 439 (2003)
MICHAEL L. KATZ & CARL SHAPIRO, *Network Externalities, Competition and Compatibility*, 75 American Economic Review, 424 (1985)
PAUL KLEMPERER, *The Competitiveness of Markets with Switching Costs*, 18 The RAND Journal of Economics, 138--150 (1987)
NET LE, *Microsoft Europe and Switching Costs*, 27 World Competition, 567--594 (2004)
STAN J. LIEBOWITZ & STEPHEN E. MARGOLIS, *The Economics of QWERTY*, *in* History, Theory, and Policy in Essays by Stan J. Liebowitz and Stephen E. Margolis, (Peter Lewin ed., 2002)
ROBERT P. MERGES & RICHARD R. NELSON, *On the Complex Economics of Patent Scope*, 90 Colum. L. Rev., 839 (1990)
RICHARD R. NELSON, *Intellectual Property Protection for Cumulative Systems Technology*, 94 Columbia Law Review, 2674 (1994)

ANDREA OTTOLIA & DAN WIELSCH, *Mapping the Information Environment: Legal Aspects of Modularization and Digitalization*, 6 Yale Journal of Law and Technology, 174 (2004)
FRANCESCO PARISI & BEN DEPOORTER, *The Market for Intellectual Property: The Case of Complementary Oligopoly*, *in* The Economics of Copyright: Developments in Research and Analysis, (W. Gordon & R. Watt eds., 2003)
FRANCESCO PARISI, et al., *Duality in Property: Commons and Anticommons*, 25 International Review of Law and Economics, 578--591 (2005)
GIOVANNI B. RAMELLO, *Access to vs. Exclusion from Knowledge: Intellectual Property, Efficiency and Social Justice*, POLIS Working Papers, n. 100 (Department of Public Policy and Public Choice - POLIS) (November, 2007)
J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Columbia Law Review, 2432 (1994)
GEORGE B. RICHARDSON, *Economic Analysis, Public Policy and the Software Industry*, DRUID Working Paper No. 97-4 (April, 1997)
MARCO RICOLFI, *La tutela della proprietà intellettuale: fra incentivo all'innovazione e scambio ineguale*, I Rivista di diritto industriale, 511--525 (2002)
J. C. ROCHET & J. TIROLE, *Platform Competition in Two-Sided Markets*, 1 Journal of the European Economic Association, 990--1029 (2003)
J. C. ROCHET & J. TIROLE, *Two-Sided Markets: An Overview*, IDEI Toulouse working paper (March, 2004)
J. C. ROCHET & J. TIROLE, *Two-Sided Markets: A Progress Report*, 37 RAND Journal of Economics, 645--667 (2006)
STEVEN C. SALOP, *Competition and Integration Among Complements, and Network Market Structure*, 40 The Journal of Industrial Economics, 105--123 (1992)
PAMELA SAMUELSON, *Benson Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions*, 39 Emory L.J., 1025 (1990)
PAMULE SAMUELSON & S. SCOTCHMER, *The Law and Economics of Reverse Engineering*, 111 Yale Law Journal, 1575--1663 (2002)
PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, 94 Columbia Law Review, 2308--2431 (1994)
JOSEPH A. SCHUMPETER, *Capitalism, Socialism, and Democracy*, (Harper and Brothers, New York. 1942)
JEAN TIROLE, *The Theory of Industrial Organization*, (MIT Press, Cambridge: Mass. 1988)
LUIGI CARLO UBERTAZZI, *Commentario Breve alle Leggi su Proprietà Intellettuale e Concorrenza*, (CEDAM Fourth ed, Padova. 2007)
C. CHRISTIAN VON WEIZACKER, *The Cost of Substitution*, 52 Econometrica, 1085--1116 (1980)
P. J. WEISER, *Law And Information Platforms*, 1 J. Telecomm. & High Tech. L., 1 (2002)
P. J. WEISER, *The Internet, Innovation, and Intellectual Property Policy*, 103 Columbia Law Review, 534--613 (2003)
SIDNEY G. WINTER, *The Logic of Appropriability: From Schumpeter to Arrow to Teece* (September, 2006)

THE LEGAL STATUS OF SOFTWARE INTEROPERABILITY INFORMATION
A Law & Economics Analysis of Application Programming Interfaces and Communication Protocols

First paper of the dissertation project:
Software Interoperability: Issues at the Intersection between Intellectual Property and Competition Policy

Federico Morando
(federico.morando@email.it)

Ph.D. Programme in Comparative Analysis of Law, Economics and Institutions

October 14, 2009

The Interuniversity Centre for the Comparative Analysis of Law and Economics, Economics of Law, Economics of Institutions

ABSTRACT

This paper shows that interoperability *specifications* are not protected by copyright.

The paper also demonstrates that existing doubts and uncertainty concerning the legal status of software interoperability information are typically related to a poor understanding of the technical nature of software interfaces. To remedy to such a misunderstanding, the paper focuses on the distinction between interface *specifications* and *implementations* and stresses the difference between the steps needed to *access* the ideas and principles constituting an interfaces specification and the ones needed to *re-implement* a functionally equivalent interface through new software code.

Leaving interoperability specifications outside the domain of copyright protection (and outside intellectual property in general) is not only coherent with general copyright law principles, as the idea/expression dichotomy; it is also likely not to generate any significant market failures and to increase competitive pressure on software market leaders.

The results of the paper are specifically discussed with respect to the legal systems of the economically more developed areas of the world: the EU, USA and Japan. The analysis is also likely to apply (with minor adaptations) to any copyright system compliant with the Berne Convention (and hence to all the members of the WTO, since the Convention has been incorporated in the TRIPs agreement), since it is mainly based on general copyright law principles, technical arguments and economic reasoning. The role of the patent system is also discussed, finding that it may represent a (mainly potential, but already clear) threat to the balancing between incentives to innovate and the need for a dynamic system of incremental innovation (balancing that is offered by the existing legal setting, in which software patents play a relatively small role).

At the normative level, the paper does not recommend major modifications to the existing model of legal protection of software (and software interfaces), as long as it is interpreted and enforced according to the descriptive part of the work. However, it suggests that policymakers could reduce the Fear of legal actions, other forms of legal Uncertainty and several residual Doubts (FUD) by explicitly stating that interface specifications are unprotectable and freely appropriable.

TABLE OF CONTENTS

1. Introduction

In the field of software, the term interoperability describes the capability of different programs to exchange data, to read and write the same file formats, and to require services from one another.[1] Loosely speaking, interoperability is a synonym of compatibility.[2] Software interoperability is at the core of several recent antitrust cases and policy debates. In the field of competition policy, the European Microsoft case (Microsoft IV),[3] concerning the disclosure of information necessary to achieve interoperability among client and server computers,[4] is worth mentioning. Similarly popularized by mass media (as a war between Apple's iTunes and the French Parliament) is the discussion concerning interoperability between digital rights management (DRM) technologies, leading to the adoption of specific regulations in France[5] and fostering much debate in other countries.[6]

The interest concerning interoperability information comes from the fact that controlling interfaces among different pieces of software (and/or hardware) is strategically crucial. And this strategic relevance originates from technological and economic phenomena, in particular from the modular nature of software programs and various kinds of network effects.[7] In the US v. Microsoft (*Microsoft III*),[8] the US Department of Justice (DoJ) summarized this point in a very effective way, quoting a Microsoft executive's statement: "to control the APIs [Application Programming Interfaces][9] is to control the industry".[10] I will come back on the definition of APIs and other kinds of interoperability information (see § 2); the point of the DoJ, however, is clear: being able to decide which programs are compatible with a crucial platform such as Windows, and to what extent, gives the possibility of dominating the entire software industry. Moreover, it can be argued that "owning" interoperability information is a way of controlling even more than the software industry, since this control could lead to significant influence in several other fields, from information and communication technology in general, to the whole content industry.[11] Nevertheless, because of several sound reasons,[12] antitrust authorities and legislators have usually adopted an agnostic attitude with respect to the actual legal status of information needed in order to achieve interoperability.

---

[1] The Digital Millennium Copyright Act 1998 (or "DMCA"), s.1201 (f) (4), defines interoperability as "the ability of computer programs to exchange information, and of such programs mutually to use the information, which has been exchanged". This definition has been borrowed, almost word by word, from the one provided by Recitals 10-12 of the Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs (hereinafter, *Software Directive*). Recital 11: "interoperability can be defined as the ability to exchange information and mutually to use the information which has been exchanged".

[2] See below, § 2, for more rigorous definitions.

[3] With *Microsoft IV* I refer to the *Case COMP/C-3/37.792 Microsoft*, that led to the *Commission Decision of 24.03.2004 relating to a proceeding under Article 82 of the EC Treaty*. The European Court of First Instance delivered its judgement on Microsoft's Appeal on the 17th of September 2007, substantially confirming the approach of the Commission (apart from some minor issues related to the Trustee established to monitor Microsoft's compliance). I refer to this ruling as *Microsoft CFI*.

[4] Actually, among operating systems (OSs) running on these computers, in particular when the servers are operated by non-Microsoft branded OSs.

[5] *Chapitre IV* (*Mesures techniques de protection et d'information*) of the *LOI n° 2006-961 du 1er août 2006 relative au droit d'auteur et aux droits voisins dans la société de l'information* (modifying the French *code de la propriété intellectuelle*).

[6] In Italy, for instance, see the proposal of reform of the Author's Right Law (Legge 22 aprile 1941, n. 633).

[7] *See General Introduction* to this dissertation project, § 2.2.

[8] With *Microsoft III* or *Microsoft US case* I refer to the famous US antitrust case, which risked leading to the dismembering of Microsoft, following J. Jackson's ruling 87 F.Supp.2d 30 (D.D.C., 2000). In appeal, the case was vacated and remanded, with ruling 253 F.3d 34 (C.A.D.C., 2001). The case has been ended by the *Consent Decree* ratified by J. Kollar-Kotelly, with ruling 231 F.Supp.2d 144 (D.D.C., 2002). Consider also that with *J. Jackson's findings of fact* I refer to ruling 84 F.Supp.2d (D.D.C., 1999). The measures adopted with the Consent Decree will stay in place only for 5 years, unless the Court finds "*willful and systematic violations*" of the terms of the agreement.

[9] APIs (or Application Programming Interfaces) are a kind of interoperability information: see below for a definition.

[10] United States v. Microsoft, 65 F.Supp.2d (1999) 1, at p.15. Also quoted in : T. A. PIRAINO, JR., *Identifying Monopolists' Illegal Conduct Under the Sherman Act*, 75 New York University Law Review, 809 (2000), pp. 888—889; P. J. WEISER, *The Internet, Innovation, and Intellectual Property Policy*, 103 Columbia Law Review, 534--613 (2003).

[11] See, among many, BRIAN FITZGERALD, *Intellectual Property Rights in Digital Architecture (Including Software): The Question of Digital Diversity*, 23 European Intellectual Property Review, 121--127 (2001).

[12] In fields which are subject to rapid technological development, the law should be as general as possible and based on general principles, in order not to be made obsolete by the fast pace of technology; moreover, the UE Commission and other communitarian authorities and courts do not have direct jurisdiction in the majority of Intellectual property rights related issues, which essentially remain a national competence.

An example of this agnostic attitude is the recent Decision of the EU Commission in the Microsoft (IV) case.[13] In that Decision, the Commission imposed on Microsoft the disclosure of "interoperability information" for any undertaking interested in developing certain kinds of workgroup server operating systems.[14] This information has to be made available "on reasonable and non-discriminatory terms."[15] At the same time, the Decision stated that "*[t]o the extent that any of this interface information might be protected by intellectual property in the European Economic Area, Microsoft would be entitled to reasonable remuneration*"[16], without providing any guidance concerning the cases in which such a protection may indeed exist[17]. This agnostic attitude is even more evident in the Court of First Instance's ruling:

> "Although the parties devoted lengthy argument, both in their written pleadings and at the hearing, to the question of the intellectual property rights which cover Microsoft's communication protocols or the specifications of those protocols, the Court considers that there is no need to decide that question in order to resolve the present case."[18]

The Consent Decree which brought to an end (at least for the moment) the main chapter of the US Microsoft antitrust saga (*Microsoft III*)[19] adopted wording that is similar to that of the Commission:

> "Microsoft shall disclose […] for the sole purpose of interoperating with a Windows Operating System Product […] the APIs and related Documentation that are used by Microsoft Middleware to interoperate with a Windows Operating System Product."[20] And "Microsoft shall offer to license […] any intellectual property rights owned or licensable by Microsoft that are required to exercise any of the options or alternatives expressly provided to them under this Final Judgment." But "[b]eyond the express terms of any license granted by Microsoft pursuant to this section, *this Final Judgment does not, directly or by implication, estoppel or otherwise, confer any rights, licenses, covenants or immunities with regard to any Microsoft intellectual property to anyone*."[21]

From previous quotations it is clear that – when antitrust authorities decide to mandate the disclosure/licensing of interoperability information – the existence and the kind of legal protection concerning that information is relevant, at least in order to determine if the disclosure may be onerous or not (evidently a quite substantial aspect of these kind of decisions)[22]. Moreover, and apart from its antitrust relevance, the protection of interoperability information that may be offered by intellectual property rights could have several effects, which are different from the ones arising from pure trade secret. For instance, intellectual property rights would allow an injunction against any use of these pieces of information,

---

[13] See supra note 3.

[14] Article 5 (a) of the Commission Decision in *Microsoft IV* (see *supra note* 3).

[15] *Ibidem.*

[16] Press Release IP/04/382, Brussels, 24 March 2004 (emphasis added).

[17] This attitude of the Commission is perfectly consistent with the competences of the European Union, which leave to Member States the bulk of the intellectual property related legislation. However, the technical nature of the subject matter and the difficulties in coordinating IP law and antitrust law (not to mention the fact that trade secrets may be considered as quasi-property rights in some European countries, including Italy, after the recent modifications to the Industrial Property Code, introduced by the Decreto Legislativo 10 febbraio 2005, n. 30) make this formally unimpeachable attitude of the EU Commission practically problematic.

[18] *Microsoft CFI* (see *supra* note 3), § 283. The Court also explains (§ 313) that this agnosticism is possible, since "there is no need to decide whether Microsoft's conduct constitutes a refusal to license intellectual property rights to a third party, or whether trade secrets merit the same degree of protection as intellectual property rights, since the strict criteria against which such a refusal may be found to constitute an abuse of a dominant position within the meaning of Article 82 EC are in any event satisfied in the [European Microsoft] case".

[19] *United States District Court for the District of Columbia, Civil Action No. 98-1233 (Ckk), State of New York, et Al. v. Microsoft Corporation*, publ. as 231 F.Supp.2d 144 (D.D.C., 2002) (hereafter, *Consent Decree*). The measures adopted with the *Consent Decree* will stay in place only for 5 years, unless the Court will find "*willful and systematic violations*" of the terms of the agreement.

[20] Section III.D of the *Consent Decree* (see supra note 19).

[21] Section III.I of the *Consent Decree* (see supra note 19) (emphasis added).

[22] Some commentators (and an old jurisprudence) actually argue that the existence of intellectual property rights should offer a complete shield against antitrust infringement. On this point, I remand to JOSEF DREXL, *IMS Health and Trinko - Antitrust Placebo for Consumers Instead of Sound Economics in Refusal-to-Deal Cases*, 35 International Review of Intellectual Property and Competition Law, 788--808 (2004). The author – who criticizes this approach – defines "inherency doctrine" the approach, "according to which intellectual property law defines the scope of protection and competition law must not interfere". Drexl confronts this approach with the "theory of complementarity," according to which "intellectual property law and competition law pursue identical goals. Both fields of law are designed to promote competition and innovation." Under this theory, "the intervention of competition laws apparently has to depend on the effects of a given IP right and its exercise on the market." (In the third paper of this dissertation, it will become apparent that I favour the complementarity approach.)

independently of the fairness of the means adopted to acquire them, potentially voiding reverse engineering of much of its utility in the field of software.[23]

Despite the relevance of interoperability information in case law and policy debates and the richness of the literature about the legal protection of software,[24] only a relatively small portion of this literature explicitly deals with the specific problem of the legal status of software interfaces.[25] In 1989 Cornish observed that, at the time, there was "still considerable misunderstanding about interfaces and access protocols, and therefore about how they should be characterised in relation to copyright concepts". [26] Unfortunately, misunderstanding remains widespread today and no clear-cut solution concerning the legal status of these pieces of code seems to have been reached: according to Välimäki, "[i]t hasn't been, and still isn't, so clear that some form of intellectual property can really cover interoperability information". [27] Rotenberg even argues that "interfaces are neither completely private, nor completely public property, but something in between." [28] Even Parasidis, in one of the most specific articles about this topic,[29] offers a detailed survey of US case law, concluding that "the extent of copyright protection afforded to APIs varies considerably depending on the specifics of the underlying computer program and the nature and function of the APIs with respect to that program".[30]

Making some clarity about the legal status of interoperability information – or, at least, understanding the reasons of the apparent lack of clarity in this field – is thus the first goal of this dissertation and the main purpose of the paper at hand. However, before concluding this introduction and despite the fact that I will not analyze this problem in the paper at hand, let me also mention that software programs can be seen as authentic "engines of free expression," so that interoperability-related issues have significant implications in terms of freedom of speech and similar issues of constitutional relevance.[31]

---

[23] The paper will discuss in detail the scope and broadness of fair use defences and specific exceptions to copyright related to interoperability. (Reverse engineering will be specifically discussed in the second paper of this dissertation.)

[24] Some of the authors having written about this topic include Gordon, Landes, Posner, Gemignani, Ginsburg, Menell, Reichman, P. Samuelson and Liebowitz, just to quote a few of them.

[25] Among the ones explicitly dealing with interfaces, interconnections, interoperability and APIs, see GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè first ed, Milano. 1996); N. T. NIKOLINAKOS, *The New Legal Framework for Digital Gateways – the Complementary Nature of Competition Law and Sector-specific Regulation*, 9 European Competition Law Review, 408--414 (2000); C. R. MCMANIS, *Taking Trips on the Information Superhighway: International Intellectual Property Protection and Emerging Computer Technology*, 41 Villanova Law Review, 207 (1996).

[26] W. R. CORNISH, *Inter-operable Systems and Copyright*, 11 European Intellectual Property Review, 391--393 (1989).

[27] MIKKO VÄLIMÄKI, *Software Interoperability and Intellectual Property Policy in Europe*, 3 European Review of Political Technologies, 1--11 (2005). See also MIKKO VÄLIMÄKI & VILLE OKSANEN, *Patents on Compatibility Standards and Open Source – Do Patent Law Exceptions and Royalty-Free Requirements Make Sense?*, 2 SCRIPT-ed, (2005).

[28] BORIS ROTENBERG, *The Legal Regulation of Software Interoperability in the EU*, NYU School of Law, Jean Monnet Working Paper 07/05 (2005).

[29] EFTHIMIOS PARASIDIS, *A Sum Greater than Its Parts? Copyright Protection for Application Program Interfaces*, 14 Texas Intellectual Property Law Journal, 59 (2005).

[30] Id., p. 89.

[31] Let me sketch in this footnote some lines of reasoning that I will completely ignore throughout this paper, but that I consider relevant in order to address the issue of software interoperability, especially because sometimes the purely law&economics based reasoning does not lead to unquestionable results. In particular, I want to summarize the point of view of Boris Rotenberg (Rotenberg, *Regulation of Software Interoperability*), who approached the problem of software interoperability from a peculiar perspective. The author started his reasoning from the well known decomposition of networks in three different layers (the physical one, the "logical" or "code" one and the content one), highlighting that the most relevant bottlenecks are likely to lie in logical layer. In fact, it is the layer, "which is responsible for filtering and channeling information to the users", while "abundant bandwidth and vast increases in processing power […] are rendering access licences as well as content regulation and monitoring increasingly suspect from a constitutional point of view". Hence, Rotenberg argues, "one compelling way to redefine 'interoperability' and third party access is through the lens of the fundamental right to freedom of expression (Art.10 ECHR [European Convention on Human Rights]), and the implicit right to non-discrimination. Approaching software regulation from the viewpoint of fundamental rights forces us to acknowledge software's unique hybrid (or dual) nature as both a means for expression, and expression in its own right. The above approach might shed new light on the question whether European law strikes the right balance between granting copyright to software writers, and enabling expression in the form of software and otherwise. Overbroad copyrights will not only silence competing software programs. Indeed, the danger exists that this same copyright enables the platform owner to have extensive control on complementary expression in the form of software code, and in the form of digital content. Eventually, this approach makes us realise that the balance struck by the law ought to be about more than just software innovation."

Rotenberg acknowledges that "[i]t has been argued that software code is not about making a point, particularly not a political one", but he refutes this approach, arguing that "[s]oftware, whether open or closed, is more than just bits and bytes. It determines which programs can be run, it empowers some speakers and can exclude others, and helps to determine a specific society's culture. To be sure, the power to construct and control channels of communication through law is a most serious *political* question in the digital era. On one side the school of thought that believes information as the basic building block of knowledge should (and wants to) be free. On the other side stands the idea that in a market economy, value added to raw information has been and inevitably will

1.1. Plan of the paper

Before describing the plan of the paper at hand, it should be noticed that this paper presuppose a general understanding of the basic economic insights concerning the workings of the software industry. A synthetic summary about these topics (and several references) are provided in the General Introduction to the dissertation, briefly touching several quite well-known (but crucial) points: network effects, learning costs, and different approaches to incremental innovation.[32] Some further economic issues will be discussed in section 8 (see below), after having described the legal background of field of analysis.

The present article proceeds in several sections.

Section 2 offers an introduction to some relevant technical problems and proposes some definitions[33] concerning the basic and fundamental technical "objects" discussed in the paper (APIs, communication protocols and similar interfaces). Since achieving interoperability with an existing software, established in the market, is frequently one, if not the main, goal of a decompilation project, this section will also explicitly address the problem of reverse engineering (from the legal point of view[34] and from the point of view of technical and economic feasibility).[35] However, I will consider decompilation only as a (normally) necessary preliminary step that is required in order to access interoperability information, the use and legal status of which constitutes the focus of this work. More specific issues concerning software reverse engineering will be addressed in detail in the second paper forming this dissertation.

Section 3 defines the fundamental distinction between specification and implementation of interfaces. Summarizing, an interface specification is essentially a set of technical requirements that must be respected in order to achieve interoperability, while an implementation is the actual software code that is written respecting the rules spelled out in the specification.

The goal of sections 4, 5 and 6 is to survey the status quo concerning the legal protection of interoperability information, with a particular attention to the US and the EU legal systems[36] (and some references to Japan). Section 7 draws some preliminary conclusions and offers some comments about the insights that emerge from the survey of US case law, to be confronted with the European situation.

Section 8 provides some more economic arguments and compares them with previous findings. Here, I will try to compare the legal findings of the first part of the paper with the economic model of the software industry sketched in the General Introduction. Then I go on to express my own view concerning the economic rationale of the so called "technology copyright"[37] model of legal protection of utilitarian works (as opposed to the patent system, or to very broad interpretations of copyright leading to similar results). The main idea of this part of the paper is that (1) copyright seems to be flexible enough to generate software innovation, in particular if it is seen as a tool devised (a) to impede free riding on up-front sunk costs of first comers, but (b) with correctives devised in such a way as not to multiply the sunk costs of late comers (in other words, copyright should not allow the first comer to force followers to have higher sunk cost than

---

be commodified and sold in the market. The fierce debate over open versus proprietary code is intimately connected with this construction of identity through software." Following this approach, pluralism – and hence interoperability – becomes a value *per se*, and it is not necessarily related to the degree of innovation in software markets nor to the existence of market power or abuses (necessary conditions to trigger antitrust intervention). In fact, the author even analogizes software to a modern agora. In this setting, he observes that "[c]opyright law is often referred to as an 'engine of free expression'; it arguably incentivises creation. But software itself, in its functional capacity, is also an increasingly important engine or medium for imparting information or ideas. Problems may arise when those two 'engines' fail to push in the same direction. [...] In the software market, copyright law has two interrelated silencing effects. First, the balance struck by copyright law has the effect of silencing at least some software expression. Second, this is likely to affect other types of expression in that some software expression that is being silenced by copyright constitutes an alternative means for (non-software and software) expression. Certain players will thus acquire tremendous power over content producers."

[32] The problem of "re-use" of software has been solved in several ways: big integrated firms, like Microsoft; open standards, like what happened at the beginning of the Internet – and in part it is still happening today; open source model.

[33] To be sure, these definitions will already include some law&economics simplifications of the technical reality.

[34] In particular, the European *Software Directive* provides a decompilation exception, allowing (see Art.6) to decompile a computer program *only* in order to make it interoperable with other pieces of software.

[35] Recent development concerning technical protection of software (and digital rights management techniques) and the legal protection of these technical tools are likely to rise additional problems in this field.

[36] In the field of Intellectual property rights, the European situation is still fragmented: whenever possible, I will try to highlight common patterns, but it will also be necessary to make reference to specific national rules.

[37] I will put an higher emphasis on copyright because this is the main tool of actual protection of software related innovation in all legal systems I am aware of.

those he had to pay).[38] Putting it in a different way, the first comer should have a fair chance of recovering its sunk costs of expression,[39] but no or very limited possibilities of making a strategic use of copyright to increase the cost of expression of the followers. In this setting, (2) the traditional "idea/expression dichotomy" is especially useful in addressing interoperability problems, (a) provided that one re-declines this idea in software markets under a "specification/implementation dichotomy": (b) merger, scenes a faire and fair use doctrines will be analysed as corrective tools to fine-tune copyright (under the approach of point 1-b)[40].

Section 9 explores more specific issues and describes alternative approaches to the interoperability debate existing in literature. Issues related to the limited, but significant, application of patent law to software products are also briefly discussed. Moreover, this section describes a possible economic criticism of the approach proposed in previous sections and offers some elements to rebut (or at least reduce the relevance) of these critiques.

Finally, section 10 offers some (provisional) conclusions, while section 11 sketches the main open problems that will be addressed in the second and third paper forming the dissertation at hand.

2. Interoperability information: technical definitions and law & economics simplifications

Application programming interfaces (APIs) and communications protocols (CPs) are pieces of software designed to allow interoperability[41] between different computer programs at computer or network level. In other words, APIs and CPs provide (different kinds of) "compatibility" between software programs, thus allowing different programs (that may have been realised by different producers) to communicate between them. Technically, an Application Programming Interface (API)[42] is the interface that a computer program provides in order to allow requests for services to be made of it by other pieces of software (including the exchange or sharing of data). A communications protocol (CP) is the set of standard rules for data exchanging (including signaling, authentication and error detection) over a communications channel. More concretely, APIs are functions including complex sets of arguments, but the details concerning the description and use of these functions are hidden in compiled code and are difficult to access (through reverse engineering) without a full disclosure operated by the original developer of the API.

In this paper, I will stress that APIs and similar interfaces[43] are actually composed by two different aspects (or logical layers, if one prefers). At the lowest possible level of abstraction there is an API *implementation*, which is the source and object code actually working in a PC, and which is quite clearly[44] (and for sound

[38] To be more clear, I would like to use some industrial economics results about sunk costs in the traditional economy, trying to show that it is not always necessary (nor good) to have small "monopolies", like the ones granted by patents understood in the "prospect theory" way. Theoretically, I would stress that it is not a good idea to economically describe copyright as a patent giving weak monopoly power; it is better to describe it as a tool to force late comers to sustain a level of sunk costs which is near to those of the first comer [but giving them a free ride on innovative ideas embodied in the copyrighted work, unless it is also patented/patentable]… Moreover, patents entail additional costs for subsequent innovators, but they embody several (more or less working and surely improvable) self-correcting mechanisms related to their different nature with respect to copyright (compulsory licenses, duration, etc.).

[39] In some paragraphs, I may make reference to copyright as a tool to recoup the sunk up-front cost of creating the first copy of an immaterial good: to be sure, I stress here that no legal tool could (or should) offer a guarantee of recoupment of these costs. The fact of recouping, or not, the sunk cost invested depends on the market success of the intangible goods created: what copyright does is it simply reduces the possibilities of free riding by third parties, reducing the characteristic non-excludability of information goods.

[40] These legal doctrines should also be analysed from the economic point of view, to understand if their role in the software industry remains the same they had in the traditional literature field: for instance, there are elements to lead one to think that the existence of the open source movement (which is quite "aggressive" with respect to the traditional players of the software industry) will highlight the absence of sound economic bases to prefer "non commercial" uses in the field of fair use… what matters is the effect on the initial market for the copyrighted good: the fact that alternative uses are for profit or not may be largely irrelevant (non-for-profit uses may actually be very cheap and "very open", so they may exercise a stronger competitive pressure).

[41] In the field of software, the term interoperability is used to describe the capability of different programs to exchange data, to read and write the same file formats, and to require services from one another.

[42] Also called Application Program Interface.

[43] Using the term APIs, I will frequently (and implicitly) include similar software objects, as CPs. Technically, my use of the term APIs will not be rigorous, but I'll try to show that from the legal and strategic point of view there are reasons to group a series of interfaces in the same law&economics discourse.

[44] In fact, I will highlight some court decisions, in which there was confusion between specification and implementation and such a clear (economic and social) need for free access to the specification, that the final result was a negation of copyright for the implementation itself. But these decisions are only a minority.

economic reasons should be) protected by copyright. At a higher level of abstraction there is an API *specification*, which is a "generalization" of the specification, stating only the necessary conditions to achieve interoperability, which – I will argue – should be and remain free to copy (apart from in very limited cases, in which an unfair competition claim could have some chance of being successful and/or the specification may be so innovative to be patented). Drawing a clearer dividing line between implementations and specifications will be one of the main goals of the paper.[45]

Even if these will not be discussed at length in this paper, some other kinds of interfaces may be worth mentioning. In particular, human interfaces, i.e. software (or hardware) elements a user interacts with, in order to exchange information with and request services to the computer. In the case of human interfaces, the problem of accessing the interface specification is non-existing: everybody knows (enough about) the brain and body of a human being and simply using a software gives sufficient information about the user interface. However, intellectual property rights may protect some solutions of particular ergonomic value, and this would impose relevant costs to anybody wanting to enter the same market. Moreover, even in cases in which various human interfaces would be fungible *a priori*, relevant learning costs may be involved[46], so that a given solution, gaining enough footing on the market, may acquire several advantages over time. Despite the focus of this paper on interfaces among different pieces of software, some of the principles that will be described for technical interfaces may still apply[47] to user interfaces,[48] in particular, if a given solution has become widely used in an industry.[49] In cases involving human interfaces, a distinction between specification and implementation is still useful, even if one should interpreted this dichotomy according to subject matter's specificities. On the one hand, actual images used such as icons, any arbitrary structure of the names of menu items (when not dictated by standard criteria, such as alphabetical order), and so on, may be part of the implementation. On the other hand, the fact of using the picture of a printer to print, or the choice of putting on the left the most used buttons, or of grouping under the "file" menus all actions needed to deal with program's output, are a likely part of the specification. To put it differently, all choices a cognitive psychologist could recommended, in order to have a more "ergonomic" piece of software, should be considered part of the specification.[50]

2.1. Access to interoperability information

Interoperability information may or may not be readily available: gaining access to this information is a necessary step (both conceptually and practically) in order to make any use of it. Even if this paper is focused on the *use* of interoperability information, it is useful to provide some hints concerning this (preliminary, but essential) *access* phase.[51] The second paper of the dissertation project will provide more details, in particular about accessing to interoperability information through reverse engineering.

In some cases, the original interoperability specification may be easily available. It actually happens quite frequently that needed API specifications are effortlessly and freely accessible, because software developers,

---

[45] See, in particular, § *3. API specification versus implementation.*

[46] An example are SAP transaction codes or – as in the example I mentioned (Mitel v. Iqtel case, mentioned in the Parasidis paper) – the command codes used (and learned by hearth) by technicians to program a telephone call controller.

[47] Several authors recognize similarities between cases concerning APIs and cases concerning other types of interfaces (like users' interfaces): "Unlike the reverse engineering of a platform standard, which involves the literal copying of software, the issue in Lotus was Borland's nonliteral copying of the user interface—as opposed to the literal copying of software as part of reverse engineering. Nonetheless, both scenarios involve functionally similar issues related to the viability of competing platform standards and should be evaluated under the same analytical framework." WEISER, *The Internet, Innovation, and IP Policy*, .

[48] An example may be the *scenes à faire* doctrine, see p. 48.

[49] In particular, if a firm did not try to have its Intellectual property rights enforced for a long while – maybe precisely to encourage the adoption of its technology as an industry (de facto) standard, then this firm may be prevented from claiming its Intellectual property rights at a later stage, when lock-in and path dependency have reinforced its market power. See § *4.3. Scenes à faire doctrine*. Similar issues may be relevant in cases concerning submarine patents in standard setting, but the same problem may arise after the creation of a *de facto* standard, if there was – at least – an implicit promise not to enforce Intellectual property rights related to the standard.

[50] The more problematic issues are those concerning solutions which were not "per se" (exogenously) superior, but which have endogenously became preferable, for instance because of learning effects.

[51] Existing case law frequently merges the two steps (access and re-use) or blurs the distinction between them: I will argue that this is a mistake, in particular because a fair use analysis (or equivalent exception with all their limitations, in civil law countries) may be necessary only in the first step (and only if access takes place through reverse engineering).

wanting to foster the creation of interoperable programs, willingly provide them[52]. In these cases – as long as the desire to collaborate with the producers of complementary products is genuine[53] – the access phase is not a problem (and the re-implementation phase is not likely to be very problematic either, apart from being costly, because of the need for skilled labour). Interoperability specifications, indeed, are very frequently available during the introductory phase of a new software technology. In fact, in this phase it is normal to perform true "evangelization efforts" to spread knowledge of the new technology, as much as possible, amongst independent developers, in order to reach a critical mass of complementary products and to attract consumers' attention[54]. However, it may be appropriate to mention that even firms favouring interoperability may want to do so keeping a certain degree of control over the disclosed information. The desire to keep some control may have various reasons: in particular, firms may want to favour the creation of complementary interoperable products, trying – at the same time – to impede the creation of competing (substitutive) compatible products;[55] or they may simply like to maintain the possibility of changing their mind in the future. To keep some control, a firm may – for instance – make interoperability information freely available in the monetary sense, but access may be subject to specific agreements concerning non-disclosure and imposing conditions for the use of this information (e.g. exclusivity or non-competition clauses, etc.). For this reason, also in cases in which access to interoperability information is monetarily free, it may be relevant to know what kind of rights are attached to this information (and – in particular – if there are intellectual property rights or other rights which may be exercised *erga omnes*). Moreover, in some cases, a firm may even decide not to disclose and document some APIs simply because doing so would generate the expectation, on the developers part, that these APIs will exist and be documented and supported also in future versions of a given software and this may not always be the case[56].

In some other cases, the original implementation may also be available (with or without a separate description of the abstract specification), with full comments (indirectly describing the specification). This is the case when the original source code (of the software with which one may want to achieve interoperability) is accessible (to everyone – as in the case of open source projects – or to the developer of complementary products, maybe as an effect of specific agreements). In fact, the source code of a piece of software is the program as developers wrote it: it is, somehow, the blueprint of the software project and other developers may fully read and understand it, especially if it is accompanied by the comments of the original programmers (which are normally interposed between the lines of actual code). If the commented source code of an implementation is available, the API specification may not be explicitly available, but its re-creation is, in principle, relatively easy (even though it may be costly and time consuming) and so we may directly deal with problems arising in the re-implementation stage. According to some commentators, the availability of the source code of the original implementation is actually the best possible case for developers wanting to achieve interoperability, in fact "*[d]ocumentation, by its very nature and the manner of its production, is always incomplete, inaccurate, and out-of-date when compared to the actual software itself. After all, the documentation is a*

---

[52] Economically, the willing provision of this "positive externality" benefiting other software developers is easily justified by the fact that the existence (and cheap production and sale) of new complementary products is likely to also increase the demand for the original program.

[53] In fact, it is not difficult to imagine cases in which interoperability information is only partially available, maybe with a strategic selection of the pieces of information to disclose, in order to disadvantage some actual or potential competitors (this scenario should look familiar to people being aware of the facts behind *Microsoft III* and *Microsoft IV* cases).

[54] JAAP H. SPOOR, *Copyright Protection and Reverse Engineering of Software: Implementation and Effects of the EC Directive*, 19 U. Dayton L. Rev., 1063 (1994), p. 1079 agrees: "Quite often the rightholder will be only too pleased to give the necessary information, as the availability of applications helps him in gaining market acceptance." However, the author also warns against several (probably less frequent, but still very relevant) cases in which this cooperation is not likely to take place.

[55] In other words, one may want to favour *vertical* interoperability (i.e. the existence of *complementary* compatible products), while hindering *horizontal* interoperability (i.e. the existence of *competing* compatible products). See below, § *7. Vertical and Horizontal Access; Transformative and Substitutive Uses.* Favouring vertical interoperability (also called direct interoperability), while making horizontal (indirect) interoperability as much difficult as possible, is a complex task. However, the majority of firms would be happy to achieve this result, simply because the first kind of interoperability generates complementary product (and increases consumers' willingness to pay for the original piece of software), while the second kind generates competing products (and likely reduces profits).

[56] This argument has been raised both by Microsoft and Apple in the context of APIs used by some Microsoft and Apple applications, but not available for third party developers. For a specific and detailed example, see programmer Vladimir Vukicevic's blog concerning the discovery of an undocumented APIs in Mac OS X, boosting the performance of the Mac version of Firefox 3 (and already used by Mac's browser Safari), available at http://blog.vlad1.com/2008/02/28/finding-the-os-x-turbo-button/ (last visited the 3rd of March 2008).

statement of intent and it is merely a word picture of the program, not the program itself."[57] Instead, source code describes the program as it really is. As I already hinted, programmers adopting the so-called "open source" model of software development make the original source code of their software, obviously including interfaces, freely available to everyone[58].

Finally, only the object (also called compiled, or binary) code may be available. Notice that even skilled human beings (e.g. talented professional developers) may be somehow able to read object code, but are essentially unable to understand more than a few lines of compiled code. In fact, this is the version of a program that may be directly executed by a computer and it is not structured as to be understood by human beings. Moreover, it is normally stripped down of all the useful comments, names of variables and other elements, that may facilitate the human understanding of the piece of software (because these elements are useless for the machine executing the object code). Indeed, object code is basically an unintelligible sequence of zeros and ones, that is quite incomprehensible also to software professionals.[59] In fact, it is not by chance that the "normal" case – when dealing with interoperability in law courts – is the one in which only the object code of the original software is available and direct access to interoperability information is impossible (or economically unfeasible, because of strategic behaviour of the holder). Of course, I am not implying that this is the most frequent case: even though the majority of commercial software houses keep their source code as a jealously guarded secret, it is actually quite usual for software producers (both commercial and open source) to promote interoperability, as I described above. What I wanted to mention is just that litigation is much more likely to emerge in cases, in which the original producers do not especially encourage interoperability. And this is more likely to happen in cases in which a given player – already established as a market leader – wants to keep control on some complementary markets and/or wants to prevent competition from them.[60]

In cases in which a player wants to prevent interoperability with its products, one of the most effective way to do so is to hinder access to the original interface specification, also preventing as much as possible, access to the original interface implementation, that is preventing other people from reading the original source code of one's software. In fact, it is relatively easy to reconstruct a specification, when the source code of the original implementation is readily available; however, it is very difficult to do the same thing starting from the compiled binary code. Thus, in the cases of the third group, when only the object code is available, software reverse engineering is the normal approach to try to reconstruct an approximation of the original implementation, which will be used in turn to understand the requirement of the original interoperability specification. In fact, reverse engineering "represents a remedy of last resort for obtaining information not otherwise available."[61]

2.1.1. Software reverse engineering (decompilation)

Software reverse engineering, also called *decompilation*, is a labour intensive, difficult and time-consuming process. To decompile a program means (loosely speaking, from the technical point of view) to translate it from the object (binary) code directly "understandable" (executable) only by a computer into a (relatively) "higher level" programming language, directly understandable by trained human beings (like the C programming language and its evolutions).[62] Languages are defined "high-level" or "low-level" depending on their "level of abstraction" from machine language. Simplifying, very high level programming languages (like the famous BASIC) are more similar to normal human language (with a very strict syntax and grammar), while very low level programming languages tend to decompose anything in the single steps that a computer (in general, or a specific kind of CPU) will perform to actually execute the program. We may imagine

---

[57] See, among others, A. JOHNSON-LAIRD, *Software Reverse Engineering in the Real World*, 19 University of Dayton Law Review, 843 (1994).

[58] Here I use "open source" in a broad way, encompassing all models of software developments requiring a widespread availability of the original source code (which must be freely accessible, but may or not be freely reproducible and modifiable): the so called "free software" model (promoted by the Free Software Foundation) is just one of the possible models encompassed in this broad and unselective definition.

[59] See JOHNSON-LAIRD, *Software Reverse Engineering*.

[60] See the third paper of this dissertation for more comments about that and for several references.

[61] JOHNSON-LAIRD, *Software Reverse Engineering*, . The work of Johnson-Laird is one of the most authoritative papers in this field, quoted also in the Sony v. Connectix ruling (203 F.3d 596 at 599).

[62] The expression disassembly – frequently used as a synonym of decompilation in non-technical texts – refers to the first part of this "translation", from object code to assembly language, which is a very "low level" language (normally designed for a specific "brand" of Central Processing Unit – CPU), nearer to object code.

executable object code as the programming language with the lowest level of abstraction. When the level of abstraction is too low, human beings are incapable of keeping track of the overall logic of the operations performed by the computer, so that – even if, in principle, they may learn how to read object code – they are unable of understanding it. However, an overall understanding of a computer program is clearly necessary in order to make another piece of software capable of interoperating with it. (And the same is true in order to modify a piece of software or to correct a bug.) Hence, programmers wanting to achieve interoperability with an existing piece of software need to decompile it (unless an interoperability specification is already available). This is why cases involving decompilation are, by far, the most common cases touching interoperability issues. Indeed, software reverse engineering is the standard tool, used to acquire interoperability information when a firm is not willing to disclose it for free or at reasonable prices. In other words, issues related to the legal status of APIs very often arise in law cases dealing also with reverse engineering: in fact, decompilation is frequently the first step to gain access to APIs. For this reason, if a firm wants to prevent interoperability, a way to do so could be by preventing decompilation. And for the same reason, despite the fact that this paper is not focused on decompilation, I will briefly describe the conclusions of the mainstream literature about software reverse engineering.

Case law and a rich literature[63] can be used to describe legal cases concerning software reverse engineering (decompilation) used to acquire interoperability. Indeed, in what follows, also specific issues concerning the intellectual property protection of APIs and CPs will be addressed in part using the same case law relating to reverse engineering.[64] I will discuss these issues more in the second paper of this dissertation, but I may already anticipate that I share the conclusion of part of the literature, according to which reverse engineering is – or at least should be, from a law & economics point of view – *per se* lawful.[65] That having been said, it will still be necessary to focus attention on allowed uses of the output of reverse engineering.[66] In fact, such an output may take different form. On the one hand, it may consist in an API (reconstructed) implementation, which could be considered the "raw output" of decompilation, i.e. the actually reconstructed source code. On the other hand, the output may be an API specification, which is a more elaborated result. In a specification, the originally obtained (reconstructed) implementation has been analysed further, several ideas have been made more abstract or general, and the engineers carrying out the reverse analysis have performed several additional attempts to decipher the meaning of a specific part of the compiled code.

To understand potential copyright issues arising from decompilation, it is important to know that "intermediary copies of the original software must be made"[67] in order to decompile it. In other words, in order to increase the degree of abstraction, going from the available object code to an approximated reconstruction of source code, it is necessary to write down several intermediate steps. At the same time, it is also important to notice that "to a much larger degree, [decompilation] is an additive process."[68] And that because the "programmer starts with the lowest possible level of abstraction devoid of any higher level information, and then adds personal knowledge and experience."[69] In other words, in order to increase the degree of abstraction, the developers performing reverse engineering intensively use their own know-how and write down from scratch new comments, names of variables and other elements which may be useful in order to make sense of the overall program. This is why a common feature of scholarly works authored by technologists is to describe decompilation "as a process of painstakingly attempting to understand the ideas

---

[63] *See*, also for more references: JOHNSON-LAIRD, *Software Reverse Engineering*, ; PAMULE SAMUELSON & S. SCOTCHMER, *The Law and Economics of Reverse Engineering*, 111 Yale Law Journal, 1575--1663 (2002); JOHN ABBOT, *Reverse Engineering of Software: Copyright and Interoperability*, 14 J.L. & Inf. Sci., 7 (2003). See also, in Italian, GIOVANNI GUGLIELMETTI, *Analisi e decompilazione dei programmi*, *in* La legge sul software, 152--201 (Luigi Carlo Ubertazzi ed., 1994).

[64] Some cases may directly concern software or software interfaces, but other may be related to different issues, like access codes to program third parties machineries or chips allowing hardware interoperability, that is cases in which physical chips actually embody software. As a general rule of informatics, consider that it is always possible to substitute a piece of software with a piece of hardware performing the same logical operations.

[65] *See*, for instance, JOHNSON-LAIRD, *Software Reverse Engineering*; SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*.

[66] SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*, p. 1608–1613: "The act of reverse engineering rarely, if ever, has market-destructive effects and has the benefit of transferring knowledge. Harmful effects are far more likely to result from post-reverse-engineering activities (e.g., making a competing product with know-how from an innovator's product). Because of this, it may be more sensible to regulate post-reverse-engineering activities than to regulate reverse engineering as such."

[67] Ibidem.

[68] Ibidem.

[69] Ibidem.

embodied in the object code of a computer program."[70] Moreover, the result of this activity need not be the same as the original source code. Not only comments and other non-functional elements will surely be different (because they are rewritten from scratch, having been expunged from the distributed object code, since they are useless for computers executing the code). Actually, also the reconstructed functional part of the program may be expressively quite different from the original one, since the same result in terms of object code may be obtained with different source codes. It is precisely because of the absence of comments, names of variables and similar elements in object code that, it has been argued,

> "[r]everse engineering does not lay bare a program's inner secrets. Indeed, it cannot. The inner secrets of a program, the real crown jewels, are embodied in the higher levels of abstraction material such as the source code commentary and the specification. This material never survives the process of being converted to object code. As the inner secrets of a program are not in the object code, reverse engineering cannot lay them bare."[71]

That having been said, it should be clear that reverse engineering is not a technique used by ordinary software "pirates", whose objective may be much more easily obtained directly copying the original object code. In fact, the objective of these pure free-riders is just to create digital copies of existing pieces of software. Obviously, they do not care about the inner functioning of cloned software. Hence, blatant infringers like software pirates use – at most – basic and limited reverse engineering of digital rights management to unlock anti-copy protections. This (i.e. how to obtain a working literal copy of the program) is all they need to understand, but – for the rest – they are not interested in any attempt to "*reconstruct*" an understandable "*simulacrum*" of the ideas and specifications embedded in the original source code.

In this paper, I assume that it is lawfully possible to access interoperability information through reverse engineering of software containing an API implementation (hence embodying several elements necessary to reconstruct a compatible API specification). In other words, I assume that decompilation is lawful, at least if performed for the purpose of achieving interoperability. On the one hand, there are elements to say that this assumption is reasonable – as a first approximation – both in the US[72] and in the EU[73] (and also in Japan).[74] On the other hand, I agree about the fact that this assumption could, and likely should, be discussed and qualified further. This is why the second paper of this dissertation will briefly survey the literature concerning software reverse engineering (decompilation) and address some open problems in this field.[75] For the purpose

---

[70] JOHNSON-LAIRD, *Software Reverse Engineering*. See also ABBOT, *Reverse Engineering* and additional references in SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering* .

[71] *See* JOHNSON-LAIRD, *Software Reverse Engineering*. This quotation perfectly applies to interoperability information and other technical details. More generally, other authors (e.g. PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, 94 Columbia Law Review, 2308--2431 (1994)) stressed that the "*crown jewels*" of a computer program may actually reside in its user interface, in the kind of problems that it address and solves or generally in its look and feel: in these cases, it is still true that decompilation doesn't "*lay them bare*", but this is true simply because they were already appropriable simply using the program and applying a so called "black box analysis" (i.e. looking at the program working in several different circumstances).

[72] See below the Atari (975 F.2d 832) or Sega (977 F.2d 1510) cases, with a broad application of fair use, but consider also the limitations imposed by the DMCA. See also SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*.

[73] See the second paper, also for comments concerning Art. 6 of the *Software Directive* and its explicit, but quite narrow exception. See, in general, references quoted in the second paper and, in particular, ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 1*, 22 European Intellectual Property Review, 7-16 (2000); ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 2*, 22 European Intellectual Property Review, 56-68 (2000); GUGLIELMETTI, *in* Analisi e decompilazione, and GUGLIELMETTI, *L'invenzione di software (1st ed.)*.

[74] R. MASHIMA, *Examination of the Interrelationship among Japanese I.P. Protection for Software, the Software Industry, and Keiretsu, Part I*, 82 J. Pat. & Trademark Off. Society, 33 (2000): "As far as reverse engineering to achieve interoperability is concerned, I believe that possible legal bases to allow a fair use defense exist under the current Japanese Copyright Law: : (i) the balancing of authors' rights with the benefit of the public and technology development (article 1) and (ii) an idea/expression dichotomy (article 2 and 10(3))." See also K. SUGIYAMA, *Reverse Engineering and Other Issues of Software Protection in Japan*, 11 European Intellectual Property Review, 395 (1991).

[75] For instance, I will argue that several limitations to decompilation are not normally adding incentives to create in the software industry, but are generating additional barriers to entry that are protecting dominant actors. [Sketching my reasoning, copyright is not protecting the ideas embodied in innovative programs, so that the dominant actors may freely copy them creating functional clones, and usually just studying them without recurring to their decompilation. (In fact, "reverse engineering of object code is generally so difficult, time-consuming, and resource-intensive that it is not an efficient way to develop competing but nonidentical programs"; Johnson-Laird, *Software Reverse Engineering*). However, once you are the platform at the centre of a market, you are protected by strong network effects (so that you do not fear functional clones, unless they are 100% compatible) and some sources of market power may come from the managing of interoperability, so that limiting decompilation is simply equivalent to reinforcing the normally available level of protection coming from trade secret, which is not likely to be a good economic policy, if innovative developers are not helped, while the position of dominant firms is strengthened.] Moreover, I will stress that the

of the paper at hand, it may just be useful to add that I will try to show that it is necessary to invoke fair use or specific copyright exceptions (as the ones established by the European *Software Directive*[76]) when dealing with decompilation, while this assumption is not necessarily true when dealing with the reimplementation of an otherwise know specification. To be more explicit, on the one hand, during the process of decompilation, the original work's expression must be copied (usually several times), even if the original goal is to reconstruct a (maybe unprotected) set of interoperability specifications. Hence, during this *access* (to interoperability information) phase it may be important to respect the requirements of the appropriate copyright limitation or exception.[77] On the other hand, when we consider re-implementation, we are no longer necessarily dealing with fair use. I argue (and I will try to demonstrate) that a properly performed re-implementation is actually a completely new expression of unprotected ideas.[78]

Another point concerning decompilation is worth noting. Even if decompilation is assumed to be a legally available option,[79] it is possible that decompiling a software is a very costly or even a non-economically-viable option. In most cases, the dubious economical feasibility of decompilation should not worry us: the fact that decompilation is a theoretically available option, but that it is also costly, is likely to create a market for the disclosure of this information, so that the original developer may "license" his trade secrets (using appropriate non-disclosure agreements).[80] However, there may be (less frequent) cases in which the strategic behaviour of the original developer generates a market failure, so that interoperability information that it would be socially optimal to spread (more or less freely) is actually kept secret, typically to protect a competitive advantage of a platform controller that has significant market power. An appropriate crafting of intellectual property law could reduce the likelihood of this situation, however – despite any effort made in tailoring intellectual property in a pro-competitive way – there may be technological reasons making this kind of situation possible. If this is the case, I consider this as a competition policy problem (in the sense that it is generated by a factual behaviour of the platform controller and by the existence of trade secrets, not by intellectual property rights or inefficient laws) and a very debated one in the literature.[81] For the moment, I will focus on cases in which decompilation is a feasible self-help option and it is actually performed in order to access interoperability information. If this is the case, I propose to take the conclusion of Samuelson and

---

European *Software Directive* [Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs: hereinafter, *Software Directive*] (or – at least – some interpretations of it) poses limits to decompilation, which are likely to be excessively burdensome to meet, in particular for the open source community. These limits concern the disclosure of the information collected through decompilation. Par. 2 of Art. 6 of the *Software Directive*: "The provisions of paragraph 1 [allowing decompilation for interoperability purposes] shall not permit the information obtained through its application: […] (b) to be given to others, except when necessary for the interoperability of the independently created computer program". This condition, in fact, could have been quite innocuous before the emergence of the decentralised model of open source development. In fact, for big software firms, it is possible to develop "in house" decompilation projects, having strict rules concerning disclosure of the results and not disclosing these results is normally in the interest of the decompiler, in order to maintain a competitive advantage. However, the cost of respecting the same limitations for a decentralised network of cooperating developers could be prohibitive. Hence, the effect could be the one of strengthening the dominant position of a few commercial platform controllers. Notice also, that obstacles to open source decompilation projects are not, in my opinion, a simple curiosity concerning a very specific legal norm. In fact, the most credible projects to replicate Windows APIs are conducted by open source programmers, like in the cases of the Wine and ReactOS projects (see the second paper of the dissertation). See the second paper of this dissertation project also for more details about the French LOI n° 2006-961. This law has been one of the few legislative interventions devoting some attention to the issue of the compatibility of proprietary software with open source solutions (despite the fact that some of the most revolutionary norms introduced by the French National Assembly have been significantly moderated in the final text of the law).

[76] Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs: hereinafter, *Software Directive*.

[77] For instance, it is typically necessary that the fair or exceptional use does not reduce the value of the copyrighted product. Again, it will not be possible to sell the decompiled code directly, because it is clearly a derivative work of the original program.

[78] For this reason, I believe that a firm should be allowed to sell/share/divulge the (newly implemented expression of the) ideas resulting from a decompilation procedure. In other words, decompilers should be free to use their tentatively reconstructed interoperability specification, as long as no part of the original protected expression is being copied (or as long as this copying may be justified under traditional copyright doctrines, including *scènes à faires*, merger and other applications of the idea/expression dichotomy). The fact that I think they *should* be free to do so does not mean that they are. In fact, article 6 of the European Software Directive severely limits the uses that may be made of information obtained through decompilation. See below for some more comments and the second paper for a full discussion of this issue.

[79] Of course, this assumption may be discussed, as I will do in the second paper of this dissertation, dealing with some specific open problems that I briefly mentioned (see *supra* note 75).

[80] This reasoning is expanded in the second paper of the dissertation.

[81] I will deal with some open issues related to this field in the third paper.

Scotchmer (2002)[82] as a starting point.[83] In fact, the authors reach two general conclusions about reverse engineering:

> "The first is that reverse engineering has generally been a competitively healthy way for second comers to get access to and discern the know-how embedded in an innovator's product. If reverse engineering is costly and takes time, as is usually the case, innovators will generally be protected long enough to recoup R&D expenses. […] Second, *we have found it useful to distinguish between the act of reverse engineering, which is generally performed to obtain know-how about another's product, and what a reverse engineer does with the know-how thereby obtained* (e.g., designing a competing or complementary product). The act of reverse engineering has rarely, if ever, market-destructive effects and has the benefit of transferring knowledge. Harmful effects are far more likely to result from post-reverse-engineering activities (e.g., making a competing product with know-how from an innovator's product). Because of this, it may be more sensible to regulate post-reverse-engineering activities than to regulate reverse engineering as such." [84]

As I already hinted, similar conclusions seem to be essentially shared, at least as long as the goal of decompilation is to achieve interoperability, also by the European legislators and by American Courts. In fact, the first paragraph of article 6 of the *Software Directive*[85] explicitly addresses software reverse engineering ("decompilation"),[86] authorizing reproductions of copyrighted programs

> "where reproduction of the code and translation of its form […] are indispensable to obtain the information necessary to achieve the interoperability of an *independently created* computer program with other programs" (emphasis added).[87]

Similarly, in the US, in the widely quoted case Sega v. Accolade[88] the Court "conclude[d] that where disassembly[89] is the only way to gain access to the ideas and functional elements embodied in a copyrighted computer program and where there is a legitimate reason for seeking such access, disassembly is a fair use of the copyrighted work, as a matter of law. *Our conclusion does not, of course, insulate Accolade from a claim of copyright infringement with respect to its finished products*" (emphasis added).

To conclude this long parenthesis about reverse engineering, it looks meaningful to concentrate our attention on the issue concerning the *use* of interoperability information (collected through reverse engineering or otherwise): in other words, we are back to the question concerning if and when APIs should be protected (and – in particular – to what extent). To discuss this issue, a fundamental prerequisite is a precise understanding of the difference between an API specification and its implementation.

---

[82] SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*.

[83] This conclusion seems to be shared by the majority of the law&economics literature focused on reverse engineering, frequently mentioning also the technical findings of JOHNSON-LAIRD, *Software Reverse Engineering*. For additional comments and a synthetic survey of the debate about the social desirability of rules allowing reverse engineering, see also ABBOT, *Reverse Engineering*, 18—20. Of course, these issues will be discussed further in the second paper.

[84] SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering* 1608—1613.

[85] See *supra* note 76.

[86] The decision of taking an explicit position concerning decompilation had at least two reasons: in terms of industrial policy, European authorities and software developers thought that this kind of exception could make it easier to catch-up with respect to the more advanced US industry (see SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*, at footnote 178); in terms of legal rules, European civil law countries lacked copyright limitations and exceptions with the flexibility of the fair use doctrine, so that it would have been difficult for courts to interpret copyright law in the sense of allowing decompilation for certain purposes not explicitly prescribed by the law.

[87] It is reasonable to interpret the limitations contained in Art. 6, concerning the results of decompilation, as applying to the reconstructed implementations (which are derivative works, with respect to the original implementation). Nothing in Art. 6 suggest an "expansion" of the standard level of copyright protection, encompassing ideas and processes and hence limiting the use that can be made of interoperability specifications (or of the independently produced reimplementation, which is – or should be – based on the reconstructed and unprotected specification, not on the original implementation). But this issue will be more deeply discussed in the second paper of this dissertation project.

[88] Sega v. Accolade, 977 F.2d 1510.

[89] Disassembly and decompilation are frequently used as synonyms (in non-technical statements). Disassembly is – so to speak – the first step in decompilation: the binary code is translated in the sequence of commands that are used to instruct the processor of a given computer. Additional steps are needed to arrive to an approximation of the original source code.

3. API specification *versus* implementation

In general, a specification is an accurate description of a set of requirements, to be satisfied by a certain product (or service).[90] An implementation (of a given specification) is a product respecting the criteria stated in the specification. According to the definition of a software technologist:

> "A software designer creates a specification embodying all of the ideas that constitute the program to be developed. Embedded in the specification are all of the higher levels of abstraction information. This information includes the reasons for creating the program, the requirements of time and space, and the general algorithms that must be performed by the program."[91]

The specification/implementation distinction is strictly related to the traditional distinction between form (expression) and content (ideas), which is at the core of copyright law.[92] Nevertheless – as the idea/expression dichotomy – also the specification/implementation distinction may be quite tricky,[93] so I will discuss it at length. A problem that is immediately clear to one accustomed to the issues arising at the borders between ideas and their expression, concerns the protection of the so called "internal form" or "internal expression" of a work. Or – stating the problem in a different way – the issue of non-literal copying, that is nevertheless reproducing a copyright-protected part of the structure of a work.[94] More generally, the distinction between ideas and expressions is a blurred one and – I concede – also theoretically questionable:[95] it is not by chance that the ancient Greek word *logos* has a semantic field extending well beyond the simple "word", including thought, meaning, reason, principle, but also speech and discourse (and, to be sure, logic). In fact, we cannot "express" ideas without recurring to some kind of "expression:" So where is the boundary between abstract ideas and external form of expression? No clear-cut or easy answers may be provided. However, despite these difficulties, the idea/expression dichotomy is a pillar of intellectual property, and not only a theoretical one: courts recur to this distinction in dealing with cutting-edge issues, like the protection of TV format or fictional characters. This is why I do not completely share the criticism against the use of this distinction in the field of computer programs.[96]

Let me come back to the specificities of interfaces. On the one hand, an API specification may be seen as a collection of ideas, but it is usually expressed and embodied in a manual or other document:[97] this is clearly a copyrighted work, as any physics or mathematics book, but the ideas it is describing are likely to be formulas, methods of operation and other non-copyrightable matters. On the other hand, an API implementation is the source/object code actually able to communicate with other software respecting the principles described in the API specification. This is another copyrightable work, both in its source-code and object-code (compiled) version.[98] Remember also that any piece of software is protected by copyright, but this protection does not extend to single words,[99] mathematical functions or technically determined

---

[90] See also *Microsoft CFI*, § 198—199: "specifications take the form of detailed technical documentation". "They describe, in particular, and in a very abstract manner, what functionalities are available and the rules which allow those functionalities to be called up and received." *Commission's Microsoft Decision*, § 24: "[a] specification is a description of what the software product must achieve, whereas the implementation relates to the actual code that will run on the computer" (see also § 570).

[91] JOHNSON-LAIRD, *Software Reverse Engineering*, p. 856.

[92] This principle has strong common law bases. *See* Baker v. Selden, 101 U.S. 99, 25 L.Ed. 841 (1879). It has also been incorporated in Section 102(b) of the US Copyright Act. The same principle has been codified in civil law countries, for instance in the European *Software Directive* at article 1: "Protection in accordance with this Directive shall apply to the expression in any form of a computer program. Ideas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright under this Directive."

[93] As the 2nd Circuit Court of Appeals puts it in Computer Associates Intern., Inc. v. Altai, Inc. (982 F.2d 693): "Drawing the line between idea and expression is a tricky business."

[94] There are countless works addressing this issue. A critical survey is provided by EDWARD SAMUELS, *The Idea-Expression Dichotomy in Copyright Law*, 56 Tennessee Law Review, 321 (1989).

[95] For a strong critique of (certain uses of) the dichotomy, *see* Id.. See, in particular, pp. 355—371 for an application to computer related issues (and the following pages for a general critique of the idea/expression dichotomy).

[96] A criticism that may be found, for instance, in Id..

[97] The specification may actually be contained in the comments included in the original source code, that is in lines of code which are not read by the computer, but are included in the software code as useful notes for current and future developers.

[98] The possibility of copyrighting object code has been one of the first issues tackled by US courts in adapting copyright to this new subject matter. Protectability has been established since the early Eighties (starting with Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 3d Cir. 1983) and it is now recognized also by article 10(1) TRIPs. For further details and references, see footnote 17 of the second paper.

[99] Or to "short phrases", under the merger doctrine or simply as a direct application of the idea/expression dichotomy.

solutions.[100] Consider also, as pointed out by many authors,[101] that two programmers implementing the same specifications will not write the same source code and will generate programs with different performances. Even if copyright can surely be used to protect a specific interface (or protocol) implementation, it is not clear if copyright can be used to create a broader monopoly, *de facto* encompassing any use of an interface specification.[102] In other words, it is quite uncontested that copyright can be effectively used to protect an implementation, but the protection of a specification seems to be nearer to the domain of patent law[103] (if any protection, at all, may exist).

In theory, an API implementation can be studied in several ways to recreate an approximation of the original specification, but it is generally impossible to recreate the full specification from a given implementation.[104] In fact, as I discussed, implementations are frequently available only in the form of object (binary) code, where all programmers' comments and notes referring to the specification have been eliminated.[105] If this is the case, any re-created specification (through de-compilation) cannot copy what is missing in the actually available code, so one may argue that a new manual, obtained from a reverse engineering exercise, is not likely to violate the copyright of the original manual only because it embodies (a newly expressed subset of) its ideas.[106] Nevertheless, several more specific problems may arise: for example a given set of words or instructions may be used as a password or lock-out code,[107] or it may be technically necessary to generate programs which are similar to the original ones under several aspects. In these cases, also the original expression is (or look to have been) copied, but with a clearly utilitarian purpose and because of technical reasons: to deal with these issues, one or more of the *merger*, *scenes à faire* and *fair use doctrines* (see below) will be needed.[108]

The reason for which I stress this specification/implementation dichotomy is that I argue that it is possible to find a way to reconcile several theoretical points of view and the large majority of case law, by stressing the distinction between API specifications and API implementations.[109] Here it is important to notice that through the paper I will normally use the term *specification* as a kind of shorthand notation for the *ideas, methods and technical principles embodied in the specification*. Hence, the word specification will refer to these non-protectable technical requirements. I will call "specification document" any kind of description of this

---

[100] This should be the case at the international level, applying – for instance – article 2 ("Scope of Copyright Protection") of the Wipo Copyright Treaty of December 20, 1996: "Copyright protection extends to expressions and not to ideas, procedures, methods of operation or mathematical concepts as such." Where "as such" means that the expression of these ideas may be protected, but not the idea itself, nor the expression in such as a way that also the idea is monopolized.

[101] E.g. ROTENBERG, *Regulation of Software Interoperability*.

[102] Some commentators would actually argue that it is indeed clear that copyright does *not* create this kind of broader "monopoly".

[103] There is a growing literature concerning the patentability of software related inventions, in particular after the rejection of the EU Council's Proposal of a Software Patents Directive. See also § *9.2. May patent law (as currently applied to software) limit interoperability?*.

[104] In fact, reverse engineering does not really provide you with the original "specification" of APIs, but only – if properly done – with a specification which is likely to be similar (and compatible) with the original one. As was observed by JOHNSON-LAIRD, *Software Reverse Engineering* : "No matter how talented the reverse engineer, and no matter how much time and money is dedicated to the task, the software reverse engineering can never recreate […] the original higher levels of abstraction information contained in design documentation, specifications, or business plans. The object code form of the program is devoid of this kind of information and the reverse engineer cannot therefore recreate it." "The original data structures, complete with data fields that might be set aside for future use, can also not be revealed. These will never be used by the program as it executes, and therefore, their purpose cannot be divined." " Reverse engineering cannot determine the original design rationale. The reverse engineer can discern what a program is doing, but not the underlying reasons why it does it the way it does, or why it does it one particular way rather than another."

[105] In fact, these notes may contain several or even all the elements of the specification, but they are useful only to write, correct and update the source code, and – not being directly understandable by a computer – are neglected during the compilation of software (the process transforming the human-readable source code into machine-readable object or binary code).

[106] It is also possible that some of the developer's choices for the binary code which is decompiled are – incorrectly – perceived as part of a general specification, even if they have been simply and arbitrarily introduced in the specific implementation at hand (this fact may create some misunderstandings and the suspect of slavish copying: this is probably at the origin of the debate around fair use to generate "current interoperability", but not "future interoperability"… Features of an implementation which are not currently used as interoperability requirements would not be freely reproducible, even if they may be necessary for future interoperability. This view is strongly criticized in JOHNSON-LAIRD, *Software Reverse Engineering*.

[107] See Sony v. Connectix (203 F.3d 596) and Lexmark v. Static Control Components (387 F.3d 522).

[108] All these doctrines are common law born, but similar results are normally attained in civil law countries making reference to the general principles governing intellectual property rights and/or invoking constitutional principles.

[109] This is true, at least, for the majority of US cases, and as long as we also decompose interoperability problems into two steps: accessing to interface specifications and re-implementing them. For further details, see below, § 4.4.1.

specification. In fact, there can be an implementation/specification dichotomy only as long as the "specification" we are talking about is formed by non-protectable technical principles. On the contrary, any "specification document" is copyright protected as a technical manual. The protection of such a manual may be "thin" (as would be the case for a geographic map or other functional creations), because the freedom of expression of the author is constrained by the necessity of respecting technical principles (and the same can be said about part of the internal structure of the work). However, one may be sure that literal copying of more than a few lines of the manual (for instance, photocopying it) will result in copyright infringement.[110]

To provide an example of the usefulness of the specification/implementation dichotomy, I will use it to rephrase the conclusions of an article by Parasidis,[111] surveying US case law concerning application programming interfaces. He argues that "the extent of copyright protection afforded to APIs varies considerably depending on the specifics of the underlying computer program and the nature and function of the APIs with respect to that program." But, one aware of the specification/implementation dichotomy may simply say that it is not possible to speak about the legal protection of APIs in general, because we need to distinguish between specifications and implementations. The author also tells that "[t]o the extent APIs are not dictated by industry standards, efficiency or the need for a program to interact with a central host computer, they are likely to be afforded protection under the copyright laws. Similarly, copyright protection for APIs is warranted in circumstances where the APIs manifest original expression is not integral to the structure and organization of a software program." Alternatively, we may say that ideas and technical procedures described by API specifications are not granted protection, generally, while implementations are copyrighted, if they respect the general requirements or copyright law. "On the other hand, copyright laws do not protect instances where the structure of a program's APIs is dictated by external factors or where the structure of the APIs merges with the underlying function of the program itself. Likewise, to the extent that copying of APIs is necessary for purposes of compatibility, a claim for copyright infringement is not likely to be upheld." Once again, elements dictated by API specifications are not protected by copyright, because they expression ultimately merges with their functional purpose. Finally, Parasidis argues that "[i]n situations where copying of APIs may be the basis for a claim of copyright infringement, the fair use doctrine may be utilized as a defense. For these cases, a thorough examination of the nature of the APIs, in relation to the underlying computer program, must be the focus of the analysis of the statutory factors which form the basis of the fair use defense." That is to say, literal copying dictated by an existing or reconstructed interoperability specification is not likely to be considered an infringement. Even if a court decided that we are dealing with copying of protected expression, at least a fair use defence would likely be available, if a certain amount of "copying" has been dictated by the respect of a common specification (and not by straightforward appropriation of an existing implementation). But – regarding the latter point – see below the discussion concerning the usefulness of the fair use doctrine in dealing with software interfaces.

In a few lines, I will go on to provide some real world examples, taken from case law, in order to better clarify how the dichotomy applies in practice. About the role of fair use in dealing with the legal protection of interfaces, here I can just anticipate that the usefulness of this doctrine is indeed very limited. In fact, I will argue that it is usually possible to solve issues related to the legal status of interfaces without recurring to fair use, with the relevant exception of the need for fair use or of a specific exception in order to allow for software decompilation.[112] I will discuss this issue in sections 4.4 and 7 and I will suggest that the solution is associated with a further distinction between the *access* (to interoperability information) phase – almost necessarily involving formal copyright violations – and the *reimplementation* phase – which should not typically need a fair use scrutiny. However, before discussing the additional access/reimplementation distinction, in the next paragraph I will review some paradigmatic decisions related to the protection of APIs through the lenses of the specification/implementation dichotomy.

3.1. Decisions of the European Commission adopting the specification/implementation dichotomy

The European Commission has (quite) explicitly adopted the distinction between interface specifications and implementations (even if the Commission is not competent in IP law issues, so that it did not draw all

[110] For an example of the typical limits that copyright can impose on the uses of any given specification document, see below the text accompanying footnote 118.

[111] PARASIDIS, *Copyright Protection for APIs*.

[112] In the second paper I will actually argue that the need for fair use (or a specific decompilation exception) derives from the *fictio iuris* of protecting object code as a literary work. See footnote 17 of the second paper and the accompanying text.

the intellectual property related consequences of this distinction). In its Decision in the Microsoft (IV) Case,[113] the Commission described interoperability policies of software vendors as follows:

> "Software vendors frequently agree to establish open interoperability standards. In this context, they usually agree on interface specifications (that is to say, specifications needed to implement compatible interfaces). Thereafter, different competing implementations compatible with the specification can be created. Such implementations may vary widely in terms of performance, security, etc. They will in principle always differ as regards their source code".[114]

The Commission went on to stress the importance of the distinction between interface specifications and implementation: "An interface specification describes what an implementation must achieve, not how it achieves it." In particular, the Commission quoted computer scientists highlighting that "a specification does not have to be concerned with details that are relevant to the implementation". Hence, it may be highly more abstract and can ignore several problems requiring paramount attention during the actual implementation phase ("e.g., memory allocation or details of most algorithms used in an actual realisation of the specification").[115] That means that it is possible to "provide interface specifications without giving access to all implementation details" and, the Commission observed, "it is common practice in the industry to do so, in particular when open interoperability standards are set."[116]

Coherently with the approach proposed in this paper and with previous quotations, the disclosure order that the Commission issued in its well-known Microsoft Decision, "concerns the interface documentation only, and not the Windows source code, as this is not necessary to achieve the development of interoperable products."[117] In other words, the Commission ordered Microsoft to release an interoperability specification ("interface documentation") and not the associated implementation ("Windows source code"). This is stressed several times by the Commission:

> "[T]his Decision does not contemplate compulsory disclosure of Windows source code as this is not necessary to achieve the development of interoperable products. The disclosure order should concerns the interface specifications only. Furthermore, as regards the subsequent use of the specifications, the specifications should also not be reproduced, adapted, arranged or altered, but should be used by third parties to *write their own specification-compliant interfaces*. In any event, to the extent that this Decision might require Microsoft to refrain from fully enforcing any of its intellectual property rights, this would be justified by the need to put an end to the abuse."[118]

Using the wording of my paper, Microsoft's competitors are supposed to write down new implementations ("their own specification-compliant interfaces"), not to copy Microsoft's implementations, nor to violate the copyright on the specification documents provided by Microsoft ("the specifications should also not be reproduced, adapted, arranged or altered"). So the Commission[119] recognizes both that specific implementations are protected by copyright and that a given specification document may also be protected (as the external expression of a maths book, as opposed to the techniques and theorems explained by it). In any case, since the Commission is not competent in the field of Intellectual property rights, which are left to

---

[113] *See supra* note 3.

[114] Recital 34 of the *Commission Decision* (*supra note* 3s.). At recital 35 the Commission also specifies that: "Specifications can in certain circumstances be accompanied by a 'reference implementation', that is to say, a source code implementing the specification. Reference implementations serve to illustrate and clarify particular points of the specification and are not suitable for direct marketing of the corresponding binary code. As such, reference implementations need to be distinguished from commercial implementations."

[115] Ibidem at recital 570. Professor Wirsing, an expert testifying for Sun Microsystems, illustrates this point by the following example. "[It] is easy to specify when a sequence of numbers is ordered: every number in the sequence is smaller or equal to its successor in the sequence. It is a lot harder to describe an algorithm for sorting a sequence of numbers and to make sure that it is correct".

[116] Ibidem at recital 571. It is also said that: "In this respect, it is also noteworthy that, under the US Communications Protocols Licensing Program, licensees are not granted access to Microsoft's source code, but to specifications of the relevant protocols". Additional comments may be found at recital 698, where the Commission notes that "there is ample scope for differentiation and innovation beyond the design of interface specifications. In his report submitted by Sun, Professor Wirsing states: 'A specification does not define all aspects of a software system, therefore many different distinct implementations of a specification are possible. These implementations may differentiate themselves by factors like ease of use, performance or scalability. Therefore specifications leave room for variation and feature enhancements in implementations.'"

[117] Press Release IP/04/382, Brussels, 24 March 2004.

[118] Recital 1004 of the *Commission Decision* (see supra note 3s.) (emphasis added).

[119] Notice that the approach of the Commission has been fully upheld in the recent ruling of the Court of First Instance (*Microsoft CFI*), ending the European Microsoft case.

the competence of national legislators, the EU authority is prudent and specifies that limitations to the full enforcement of Microsoft's Intellectual property rights could possibly be necessary to end Microsoft's abuses.[120] But the following parts of this paper will demonstrate – as the Commission seemed to suspect already – that no limitations of Microsoft's existing intellectual property rights (*strictu sensu*) are strictly needed to implement the Decision, apart from the obvious (and economically very significant) duty to release a trade secret to its competitors. However, this is not an expropriation of any form of existing property,[121] but "just" a limitation of Microsoft's freedom of movement in the market (as frequently happens to the dominant undertaking, confronted with the "special responsibilities" arising from its market position).

3.2. Implementation v. specification: some more hints from US case law

US case law provides several insights, which are coherent with the specification/implementation dichotomy that I proposed as a general category to systematize the various positions concerning the protectability of software interfaces.

In the following subsections, I will describe in some detail a few of the technologically simpler cases, because these applications will make clear that transposing the dichotomy in actual decisions is possible and coherent with the present findings of the courts.

3.2.1. E.F. Johnson Co. v. Uniden Corp.

The case E.F. Johnson Co. v. Uniden Corp.[122] concerned an innovative mobile radio communication system ("logic trunked radio" or LTR system). The defendant entered into the market for compatible radios and repeaters, realizing a software which had to be compatible with the one managing plaintiff's devices. Analysing the radios of the defendant,[123] the plaintiff's engineers found (as the court confirmed) several suspect analogies (including likely copied portions of code).[124] One could be tempted to list this case among the ones "indicating that copyright protection for APIs is warranted". However – keeping in mind the previously discussed dichotomy – it will be easy to understand that it is actually a case which is perfectly coherent with the protection of API implementations against literal copying, while allowing almost complete freedom to reproduce elements dictated by API specifications.

As an example of a typical (even thought quite minimal) interoperability requirement, I quote a part of the court's findings:

> "Due to the fact that Uniden designed its radio to be compatible with EFJ's LTR system, both parties acknowledge that some similarities in software design were inevitable. The Court finds that in order to make its radios compatible with LTR repeaters Uniden was required to copy the 'Barker code' found in the copyrighted EFJ program. A 'Barker code' is a pattern of ones and zeroes alternated in a prepatterned sequence. Both the sending and receiving units must identify the Barker code in order for communication to be established. The EFJ Barker code is numerically depicted as 1011000. In order to make its radios compatible, Uniden was required to and did copy this aspect of the EFJ program."

Therefore, as it frequently happens, an interoperability specification dictated the reproduction of some apparently arbitrary expressions (the "Barker code") in any compliant implementation. To deal with this problem, the Court summarized, in this way, the test created by the Third and Ninth Circuits[125] to analyze copyright infringement in the field of software:

---

[120] This is coherent with the Commission's attitude in Magill and IMS-Health cases, where the Commission was plainly sceptical about the opportunity of granting Intellectual property rights in the fields at hand, but it refrained from discussing this matter (outside of its jurisdiction).

[121] There is a significant difference between intellectual property rights (properly said) and trade secrets: despite the fact that both legal institutions may be used to protect intangible assets, intellectual property rights may be used *erga omnes* and their subject matters are explicitly defined by the law, while trade secret may protect any kind of information and ideas, but essentially only against various forms of unfair appropriation.

[122] 623 F. Supp. 1485 (D. Minn. 1985). Being on of the first cases dealing with interoperability issues, the Court decision is accompanied by a whole range of definitions and technical details (including a glossary and several detailed explanations and examples).

[123] Incidentally, the simple act of analysing this product (through decompilation) would probably be formally prohibited by the narrow interoperability-specific exception of Art. 6 of the European *Software Directive*.

[124] Software code was embodied in physical devices, but this is not relevant (as the parts conceded and the Court correctly recognized).

[125] See Apple Computer v. Franklin, 714 F.2d at 1251 and Apple Computer v. Formula, 725 F.2d at 525.

> "whether other programs can be written which perform the same function as the copyrighted program. If other programs can be written or created which perform the same function as the copyrighted program, then that program is an expression of the idea and hence copyrightable. If a specific program, even if previously copyrighted, is the only and essential means of accomplishing a given task, their later use by another does not amount to infringement."[126]

To the same effect, the Court also quoted the CONTU[127] final report:

> "In the computer context [...] when specific instructions, even though previously copyrighted, are the only and essential means of accomplishing a given task, their later use by another will not amount to an infringement."[128]

That having said, however, the Court specified also that some specific parts of the expression, like the already mentioned "Barker word", were "of necessity identical in both codes" (in order to achieve interoperability), but other parts of the software needed not to be identical (and hence could not be legitimately reproduced). In particular, the Court found some identical mistakes in the two software and this was considered a very reliable proof of literal copying, which was surely technically unnecessary (and actually technically detrimental).[129] The courts clearly summarized its conclusion saying that

> "the mere fact that defendant set out with the objective of creating an LTR-compatible radio does not, without more, excuse its copying of plaintiff's code. The Court finds that copying plaintiff's code was not the only and essential means of creating an LTR-compatible software program. Defendant was required to copy plaintiff's Barker word, as discussed above. Virtually all other aspects of defendant's program could have been independently created, however, without violence to defendant's compatibility objective. Defendant has reproduced the expression, not merely the idea of plaintiff's copyrighted program."

A contrario, reproducing ideas, and not expression, would be legitimate and also reproducing expression would be allowed, in cases in which this is "the only and essential means" of creating a compatible product.

3.2.2. CMAX/Cleveland v. UCR

Another example may come from CMAX/Cleveland v. UCR.[130] This case is explicitely listed by Parasidis as among the "Cases Indicating Copyright Protection for APIs is Warranted".[131] However, I argue that this is just a case where the defendant had literally copied the expression of the plaintiff's interface implementation.

UCR operated in the rent-to-own business: it rented various consumer audio/visual and other products to final customers. Computermax (CMAX) was a computer software company, producing the RMAX System, designed to enables rent-to-own companies to input, store, process and retrieve information related to their business (inventory, rental agreement, accounting information and so on): the RMAX System was comprised of two related software packages, the RMAX Remote Store System (client part) and the RMAX Host System (server part). The Plaintiff, CMAX, provided its important customer UCR with the source code of its copyrighted programs, in order to improve the cooperation and communication regarding problems and request improvements. Admittedly using the work of programmers of limited experience, and continuously referring to the actual implementation of the RMAX client to solve programming problems, UCR developed its own version of the RMAX system (UCR developers admittedly copied the entire design of the software, including screen display, reports, menus, file format and so on).

In this case, it is evident that the interoperability implementation had actually been copied, along with several other elements of the RMAX client. As the Court found:

---

[126] It should be said that the Johnson v. Uniden court seems to interpret the "only and essential means" requirement in the sense of saying that a given mean is the only and essential one not in absolute terms, but at a given level of efficiency.

[127] The National Commission on New Technological Uses of Copyrighted Works (CONTU) had been established by the US Congress to survey issues concerning the application of copyright to new technologies.

[128] 623 F. Supp. 1485 at 1502, quoting National Commission on New Technological Uses of Copyrighted Works, Final Report 20 (1979) (*CONTU Report*).

[129] 623 F. Supp. 1485 at 1496: "The Uniden select call prohibit feature incorporates the same error found in version 3.0 of the EFJ software. For this comity of errors the Court can conceive of only two plausible explanations: one, that EFJ and Uniden engineers independently committed the same inadvertences; or two, that Uniden engineers unknowingly wrote the error into their code when copying the EFJ code. The Court finds the latter explanation to be the likelier one."

[130] CMAX/Cleveland v. UCR, 804 F.Supp. 337.

[131] PARASIDIS, *Copyright Protection for APIs*, p. 74—76.

> "The data elements appear in the respective files of both systems in identical order, and in most instances, are the same length. The ordering method utilized in the RMAX Remote Store System, however, is not alphabetical or otherwise systematic, nor is it functionally significant in any respect. Thus, there is no reason why the UCR System should have the same data elements in the same order that the RMAX System does, absent copying."[132]

In fact, these similarities were not dictated by any interoperability requirement, since "some file layouts in the UCR System have a number of fields that do not appear in RMAX Remote Store System files",[133] so that it was clear (to the Court) that interoperability between the client and server systems did not require a strictly given number of, or a structure for these fields. In fact, the Court may even have been wrong – technically speaking – in concluding that some specific instances of copyright were not needed to achieve interoperability. The point is that evidence of parasitic copying was so clear and abundant that some legitimate copying could not have excused them. Overall, the Court did not give any sign of evaluating that copyright prevented the defendant from creating a piece of software interoperable with that of the plaintiff, it just and simply verified that, in this specific case, the interoperable product had been realized by unskilled programmers, using the source code of the original piece of software as a continuous reference, hence creating a clearly derivative work. Banalizing, the defendant's developers did not analyze the plaintiff's software in order to reconstruct a specification document to be used as a technical constraint in developing their own piece of software. Instead, they just tried to realize a piece of software which was as similar as possible to the one they had access to, just rephrasing (or even copying) the solutions adopted by the plaintiff when they were not sure about something.

3.2.3. How to distinguish between copying ideas and expressions?

The aforementioned cases also offer some hints about practical rules used to distinguish between the copying of ideas and Expression. In order to do that, let me borrow something from the literature on the idea/expression dichotomy.[134] From that literature, it is evident that the majority of cases decided on the basis of the dichotomy "really do[es] not hinge ultimately upon a characterization of a work in isolation, but instead involve a close comparison of a copyrighted work with an allegedly infringing work".[135] In fact, it is well known that there is no preliminary test of "copyrightability" of intellectual creations, so that the possibility of using copyright to protect a given work is always evaluated *ex post* by the legal system. And that typically happens in the context of a litigation, where what is being decided is ultimately the scope of the excluding power granted by intellectual property. That implies that it is not necessary to apply the idea/expression dichotomy as an abstract test (to determine if something could be protected or not); to the contrary, it is sufficient to apply the dichotomy just to determine if something that has been copied was a freely appropriable idea or protected expression. Hence, "whenever possible, [the dichotomy] should be applied at the infringement stage, in order to allow the comparison of the copyrighted work with an allegedly infringing work, rather than applied at the threshold stage to a copyrighted work in isolation".[136] In fact, applying the dichotomy as an abstract test, we risk facing prohibitively difficult issues and we could be tempted to say that interface specifications are not protectable at all, while interface implementations are always protected. That is not the case and thinking in these terms is misleading. Hence, violation will depend upon the typical test of substantial similarity, but here the idea/expression dichotomy will come into play, preventing similarities dictated by technical principles, ideas and methods from leading to a finding of substantial similarity.[137] In other words, the advantage of applying the dichotomy at a later stage, in the

---

[132] 804 F.Supp. 337, recital 83. The Court is actually reporting a specific example of this evidence of copying, that I want to repeat here in order to give an intuition of the kind of evidence used to back the accusation of copyright violation (recital 82): "*The RMAX Remote Store System's Rental Agreements file contains five 'late payment' fields in consecutive order. In the RMAX file, 'LATE.1', 'LATE.2' and 'LATE.3' appear as the eighteenth, nineteenth and twentieth fields, respectively, while the 'LATE.4' and 'LATE.5' fields appear in the forty-third and forty-fourth positions. Obviously, the two latter fields were added some time after the first three 'LATE' fields. The UCR System's Rental Agreement file utilizes the exact same ordering method. […] The Court agrees with [the Plaintiff's expert testimony's] conclusion that a programmer creating a new file from scratch logically would have placed these five data files together, rather than placing three together, inserting twenty-three unrelated fields, and then adding another two. The only reasonable explanation for this similarity is that the UCR System's programmers copied the RMAX Rental Agreements file.*"

[133] 804 F.Supp. 337, recital 83.

[134] In particular, from SAMUELS, *The Idea-Expression Dichotomy*, .

[135] Id., p. 419.

[136] Id., p. 462.

[137] See also § *4.1.2. Adoption of the "three-step test" of Computer Associates v. Altai.*

context of a substantial similarity test, is that we do not need to decide in abstract if any given word is part of an unprotected idea or of the protected expression. Instead, we are allowed to focus on the existing similarities and we may just evaluate if they are dictated by the underlying ideas or not.

For instance, EFJ v. Uniden makes evident that a typical way of detecting the copying of the expression is simply to find similar bugs in two pieces of software.[138] Obviously technically superfluous instructions, declarations of variables and the like may give similar hints[139] (in some cases, some odd programming choices may even have been expressively introduced in the code in order to provide a proof of copying, without any other meaningful use).[140] In general, finding statistically unlikely similarities, in cases in which the same technical problem had several different solutions (without one being more efficient or appropriate than the others)[141], provides a relevant evidence of copying of protected expression. An especially telling example may be taken, again, from EFJ v. Uniden. In that case, "miscellaneous evidence of copying abound[ed]", but one instance is especially interesting. Several (counterintuitive) programming choices of the plaintiff had been taken for efficiency reasons related to the specific characteristic of the Intel microprocessor installed in its radios; without any apparent reasons, the defendant (using a Hitachi microprocessor, with different technical characteristics) took exactly the same choices (which were suboptimal on its own, different microprocessor).

This kind of evidence, where similarities in the expression are not only unnecessary, but even counterproductive for the late comer, are far more common than one may expect in cases where original expression is copied. In fact, the main point of copying someone else expression is precisely to save on development costs; hence copiers will typically be quite sloppy in their copying activity. In principle, it would surely be possible to fully understand someone else code and reproduce its structure – maybe in such a way that could violate the original copyright – without leaving much evidence of the copying. However, the cost of this kind of copying – unless it is done by illegally appropriating secret information and/or stealing workers[142] – is likely to be so high that it does not pose any major competitive threat to the first comer. To be sure, some kind of detection errors are unavoidable. However, the risk of "false negatives" in case of very "high quality" copying (i.e. copying that is possibly reproducing some protectable expression, but that is performed very near to the idea/expression "border") is a risk that entails much less costs than the opposite "false positive" case. In that case, the legal system would risk granting patent-like protection through a tool, copyright, that does not incorporate the appropriate pro-competitive corrective elements.[143] In fact, as I will try to clarify (in § 8 and in the second paper) the economic meaning of forbidding the copying of expression is to make sure that late comers did sustain development costs, which are similar to the ones borne by early comers. What we want to avoid are cost savings based on copying. If there is evidence that some copying has been performed to save on costs (or, equally, just because of laziness), then we do not want to allow this copying. However, if – in the context of an independently written software – some apparent "copying" is likely to have been dictated by technical reasons, in order to achieve compatibility, then there is no reason (as I will try to demonstrate in § 8) not to allow it. Actually, I will try to demonstrate that forbidding similarities in expression dictated by technical reasons would very likely reduce social welfare.

In fact, the favour that the so-called "clean room" process of software reverse engineering finds in front of US courts confirms that American judges share the previous reasoning. In the clean room process, two

---

[138] Errors are relatively frequent in software code (because several of them do not impede or hinder the correct working of the program, if not in very specific situations), so that case law is rich of examples of copying activities detected (also) using copied mistakes.

[139] Again in EFJ v. Uniden, the Court found that "[p]recisely the same three superfluous instructions are found in the Uniden [defendant] program, in precisely the same location [with respect to what happened in the plaintiff's software]".

[140] Notice that, in principle, some of these arbitrarily introduced pieces of code could have been inserted as a kind of "time bomb" against unofficial compatible products: in other words, they may become necessary for future interoperability (see JOHNSON-LAIRD, *Software Reverse Engineering,* ; RICHARD H. STERN, *Reverse Engineering for Future Compatibility*, 1 European Intellectual Property Review, 175--180 (1994)). Hence, it is actually possible to imagine cases in which some bugs are knowingly copied and introduced in a new program for compatibility reasons. However, such a decision could be testified – for instance – by specific notes in the source code of the allegedly copied reimplementation: hopefully, no programmer would willingly copy a bug without adding a comment explaining that the following lines are not completely correct and why they are not.

[141] For instance, in EFJ v. Uniden, a matrix that could be structured in 32 different ways had been structured exactly in the same way in two types of software (this fact was not considered in itself, but in the context of several other suspect similarities).

[142] All cases in which there are legal tool to remedy to possible market failures which are more appropriate than copyright and that entail less risks of market distortive effects.

[143] These elements include the preliminary scrutiny of novelty and existence of an "inventive step", disclosure obligations and so on. *See*, in particular, GUSTAVO GHIDINI, *Profili evolutivi del diritto industriale. Proprietà intellettuale e concorrenza* (Giuffrè, Milano. 2001) or GUSTAVO GHIDINI, *Intellectual Property and Competition Law. The Innovation Nexus* (Edward Elgar. 2006).

separate teams of engineers perform the reverse engineering work frequently needed to access interoperability information and the reimplementation work needed to create a new specification-compliant interface.[144] The main point of adopting such a complex procedure is to avoid any unnecessary copying of expression, but its main economic effect is that the development cost of late comers is likely to be the same as that of earlier developers (since they start their work from scratch, with only the guide of a specification document deprived of any unnecessary implementation-specific expression).

4. Investigating the legal status of interoperability information: US case law and doctrines

The literature seems to be completely consistent in stressing that US case law concerning both reverse engineering and interoperability issues in general is far richer and more consistent than that established from EU Cases.[145] That is consistent with the fact that the United States saw the fastest development of the software industry and represented for years (and possibly still today) the biggest market for software goods. Fortunately, E. Derclaye showed[146] that there are fundamental similarities between the statutory texts concerning software protection on both sides of the Atlantic Ocean, which are "reflect[ing] the same policies". Thus, she concluded her analysis confirming that the US case law is "readily applicable" in the EU on several points, including "the general idea/expression principle", the interpretation of which is crucial for the paper at hand. Obviously, one needs to be very careful in applying principles derived from the US common law system to civil law countries, but the analysis offered by E. Derclaye suggests that starting my investigation from the US case law could offer several useful insights.

4.1. The protection against copying of non-literal elements

The starting point to understand copyright protection of interfaces is a deeper understanding of the copyright protection of software against non-literal reproductions. To analyze this issue, I will start discussing the "look and feel" test of the Whelan case and the three-step test of Computer Associates v. Altai.

4.1.1. The Whelan or "look and feel" test

In *Whelan,* the defendant had allegedly copied the non-literal structure of a dental lab management program and the idea/expression distinction test proposed by the Third Circuit was the following:

> "the line between idea and expression distinction may be drawn with reference to the end sought to be achieved by the work in question. In other words, the purpose or function of a utilitarian work would be the works idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea […] Where there are various means of achieving the desired purpose, then the particular means chosen is not necessary to the purpose; hence there is expression not idea."[147]

This test is (appropriately) based on the idea/expression dichotomy, but it risks being overbroadly interpreted, as happened in the Whelan case itself. In that famous case, the Court concluded that the idea of the plaintiff's program was just "*the efficient management of a dental laboratory*", invariably deciding that any kind of similarities between two software program were copying of "the program's expression", so that it becomes very easy to extend copyright protection to the structure, sequence and organization of a computer program. Even if this same test may be interpreted in a more strict way, it definitely leaves an excessive scope to the discretion of the court, increasing legal uncertainty. Moreover, protecting everything but "the idea" behind a program (in the sense of its purpose) would imply a broadness of copyright protection that would be similar to the one offered by patents, that indeed typically protect one out of several technical "means of achieving the desired purpose". Since software (at least in the US) may enjoy patent protection, but only if it is able to integrate restrictive conditions, it would be very problematic to offer an almost equivalent protection via copyright, because this would create incoherence in the intellectual property law system, at a

[144] See the second paper of this dissertation for more details.

[145] This view looks to be shared – more or less explicitly – in all the articles I quoted in the paper at hand, but see – in particular – DERCLAYE, *Software Copyright Protection -- Part 1*.

[146] See Id., and DERCLAYE, *Software Copyright Protection -- Part 2*.

[147] 797 F.2d at 1236.

systematic level.[148] Because of these reasons it should presently be undisputed that copyright cannot provide a monopoly over all the possible means to achieve the ultimate purpose of a software: not even a whole set of patents could normally achieve this goal[149]. But even offering a monopoly on a single means of achieving a desired purpose (when others are available), as suggested by the Court in *Whelan*, would transform software copyright into something that risks being equivalent of an automatically granted and almost everlasting software patent (with the only qualification that an alternative means to achieve the same end must exist)[150].

4.1.2. Adoption of the "three-step test" of Computer Associates v. Altai

Due to the aforementioned reasons amongst others, the *Whelan* or "look and feel" test has, nowadays, been supplanted by the so-called *Altai*[151] or three-step test.[152] The Second Circuit developed the three-step analysis in *Computer Associates v. Altai*,[153] based on the abstractions test, originally developed in the '30ies by Judge Learned Hand in dealing with similarities between plots and other "structural" elements of traditional literary works (plays, in particular):

> "In ascertaining substantial similarity under this approach, a court would first break down the allegedly infringed program into its constituent structural parts. Then, by examining each of these parts for such things as incorporated ideas, expression that is necessarily incidental to those ideas, and elements that are taken from the public domain, a court would then be able to sift out all non-protectable material. Left with a kernel, or possible kernels, of creative expression after following this process of elimination, the court's last step would be to compare this material with the structure of an allegedly infringing program. The result of this comparison will determine whether the protectable elements of the programs at issue are substantially similar so as to warrant a finding of infringement."[154]

The fist step of the test, called "abstraction", requires a court to "dissect the allegedly copied program's structure and isolate each level of abstraction contained within it." The two extreme levels are represented by the actual program code (at the minimum level of abstraction) and by "an articulation of the programs ultimate function" at the highest level of abstraction.[155] The second step is that of "filtration", which "entails examining the structural components at each level of abstraction to determine whether their particular inclusion at that level was 'idea' or was dictated by considerations of efficiency."[156] In other words, at this stage the merger doctrine is applied, along with *scenes à faire* doctrine (see below, § 4.3). Only in the last step, "comparison", is the issue of substantial similarity addressed, in an examination of the parts that survived the filtration step, to determine "whether the defendant copied any aspect of this protected expression, as well as an assessment of the copied portion's relative importance with respect to plaintiff's overall program."[157]

About the first step, a few comments may be useful. Computer programs may typically be decomposed in several layers of abstraction, starting with the set of individual instructions (organized in a hierarchy of modules) in the form of source code or object code. At a higher level we may find the set of the lowest level modules with their function, in which we substitute some lines of code with their common purpose; until we arrive to the set of functions of the highest level modules (like a sketched plot, chapter by chapter, for a book) and finally to the ultimate function of the entire program (not much more than the title, for literary works). To provide an intuition of the kind of "decomposition" which is needed, let me use a less technical example. Think about the problem of loading a truck with baskets of apples: let's imagine that we have to

---

[148] See GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè second ed, Milano. 1997), pp. 263—267, but also pp. 243—247. In particular: "L'esigenza di coordinamento con i brevetti comporta che la tutela d'autore sul software non possa avere un'ampiezza uguale alla tutela brevettuale e, *a fortori*, neppure evidentemente maggiore" (i.e. for reasons of systematic coherence with patent law, copyright cannot have the same scope of patent protection and, *a fortiori*, cannot have a broader scope).

[149] *Sic* Id.: "Iniziando da questo profilo [*see supra* note 148], si spiega perché il diritto d'autore non si estende fino a (tutte le alternative forme progettuali che realizzano) lo 'scopo' e (che quindi risolvono) il 'problema' alla base del programma. La monopolizzazione di tali elementi non sarebbe possibile neppure con il brevetto." (Expressing the same concept stated in the main text.)

[150] Id., pp. 247—251.

[151] 1992 WL 139364 (2nd Cir.(N.Y.)).

[152] See, among others, DERCLAYE, *Software Copyright Protection -- Part 2*.

[153] *Computer Associates v. Altai*, 982 F.2d 693.

[154] Id., p. 706.

[155] 1992 WL 139364 at 13.

[156] 1992 WL 139364 at 14.

[157] 1992 WL 139364 at 19.

program a robot to do so. At a quite low level of abstraction (but this is already very high, with respect to the actual programming of a robot!), we have to instruct the robot to "look for a basket", "take it", "look for an apple", "put the apple into the basket", "verify if the basket is full, if not, look for another apple, it it's full, put the basket in the truck" and so on… At a higher level of abstraction we simply tell to the robot: "full the basket with apples" and "put the baskets into the truck". At the highest level of abstraction, we just say: "load the truck with apples". In the field of software the "creativity" requirement associated to copyright is strictly related to the degrees of freedom available in solving a given problem via software.[158]

The decomposition step is relevant because the degrees of "expressive freedom" of authors may vary at different levels of abstraction. Hence, at some levels of abstraction we may be in a field in which developers' choices are likely to be of an expressive kind, while at other levels various kinds of technical or logical constraints may be so prevalent that expressive freedom is essentially inexistent. In principle, courts should look for ideas and expressions at any of the level of abstraction, because different elements may determinate expression both at very low and very high levels of abstraction. On the one had, choices may be limited or even technically determined at the lowest levels of abstraction, where the strict "grammar and syntax" of programming languages may determine the present way of solving a problem (or may simply leave such a limited number of possible solutions, that monopolising one would unacceptably restrict the freedom of expression of late comers). At this level, it is obvious that individual instructions cannot be copyrighted, because authors have to respect the "orthography" of a given programming language; but it is equally true that some groups of instructions may not be copyrighted because they are dictated by the grammar and syntax of a given programming language. However, it is almost impossible that – at this very low level of abstraction – an entire program could be considered as technically determined. But this simply amounts to saying that literal copying of "big enough" parts of a piece of software is prohibited by copyright. On the other hand, at very high levels of abstraction authors are just stating technical problems and possible ways to solve them, so we are clearly outside the domain of copyright law (and – at most – novel and inventive solutions could be protected with patents, if the other requisite for patent protection in a given legal system are respected). Hence, at some levels of abstraction there will be no protectable expression, while at other levels (possibly more than one) expression maybe protected, in principle. At these levels, it will be necessary to verify the existence of substantial similarities between the two pieces of software. However, before performing the actual comparison, an additional filtration step is required.

I will tell more about the three-step test after having commented on my view of the economics of technology copyright in § 8.

4.2. Merger doctrine

According (for instance and to remain in the field of software and APIs) to Baystate v. Bentley (1996)[159] under the merger doctrine, protection is denied to expression that is inseparable from the ideas (including processes or facts). Similarly, when there are "only a few means of expressing an idea", copyright protection is denied to these expressive means, in order to leave the underlying ideas in the public domain. In other words,

> [t]he doctrine's underlying principle is that '[w]hen there is essentially only one way to express an idea, the idea and its expression are inseparable and copyright is no bar to copying that expression.'[160]

It is appropriate to mention that part of the literature[161] criticized the merger doctrine as an excessively restrictive reading of the broader idea/expression dichotomy. For instance, Karjala argued that:

> The difficulty with the merger approach to idea/expression in functional works is that Baker v. Selden and section 102(b) of the Copyright Act mean much more than the merger doctrine. The systems and processes described in a copyright-protected work are unprotected no matter how many other possible systems or processes may exist to accomplish the same result and regardless of whether they accomplish that result less, equally, or more efficiently. The Court in Baker did not inquire into whether other

[158] See GUGLIELMETTI, *L'invenzione di software (2nd ed.)*, , because this would create incoherence in the intellectual property law system, at a systematic level., pp. 252- 262.

[159] Baystate v. Bentley, 946 F.Supp. 1079 (U. S. District Court, D. Massachusetts. Baystate Technologies, Inc., Plaintiff, v. Bentley Systems, Inc., Defendant. Civil Action No. 96-40196-NMG. Dec. 6, 1996).

[160] Altai, 982 F.2d 693 at 707-08, internal quotations omitted.

[161] *See*, in particular, DENNIS S. KARJALA, *Copyright Protection of Computer Documents, Reverse Engineering, and Professor Miller*, 19 University of Dayton Law Review, 975 (1994).

> accounting methods existed to accomplish the goals of Selden's system-in principle there are millions-let alone any that were better than the one at issue. The fundamental notion is that functional works not meeting the stringent requirements for a seventeen-year patent must be allowed to develop through incremental change, via the contributions of many creative persons and not just the person who first arrives at a particular stage.[162]

Here, Karjala correctly stresses that the merger doctrine should not be reversed and used to say that anything that may be expressed in more than one way deserves copyright protection. But his approach downplays the importance of the fact that a defense of independent creation does exist in the field of copyright and that the protection afforded to any kind of "expression" could be essentially limited to protection against copying, i.e. against some form of parasitism, which is much more narrow than the kind of exclusive right offered by patent law. That is particularly true in the field of software, where it is far from obvious that a given developer had access to someone else's original expression (in fact, this access is meaningful only if it is an access to the source code of someone else's software, because copying object code would almost invariably result in literal copying, that would be much easier to detect and that would need no complex evaluations to be considered an infringement).

Additionally, also other criticisms have been raised. For instance, it has been observed that:

> "the merger defense has succeeded where the total expression has been small and the purpose of the expression has been well defined." [In general, this fact may imply that] "[t]he expanding body of languages, commands, functions, and procedures, which will continue to grow as microprocessors continue to advance, means that a task or idea may be implemented in an increasing number of ways. Thus, the defense of merger should be met with some skepticism."[163]

For the purposes of the paper at hand, however, this "limit" of the doctrine should not be worrying. In fact, as long as we are dealing with interfaces and the "copying" is mediated by the reconstruction of an interoperability specification, typically only limited a limited amount of software code needs to be copied and with a precise purpose represented by interoperability, so that merger doctrine is in its more natural field of application.

If a philosophical parenthesis is allowed, not applying the merger doctrine would violate the "Lockean Proviso" – frequently recalled by Nozick and other libertarians –, concerning property (obtained by labour) on land and other resources that may be scarce: the property of one should leave "enough and as good" for others.[164] Hence, a given expression of an idea or algorithm can be protected, but not to such an extent that protecting it increases the cost of expression of other people wanting to use the same idea or algorithm. (Coherently, there must be a defense of independent creation and expressions "determined" by function cannot be protected or receive a very "thin" protection – i.e. they are protected against literal copying, if also some minor and non-functional elements are copied, but nothing more.) About the merger doctrine, let me conclude with an interesting qualification, proposed in Judge Feikens' (in part dissenting) opinion in the Lexmark v. SCC case:

> "[A]n otherwise copyrightable text can be used as a method of operation of a computer—for instance, an original, copyrightable poem could be used as a password, or a computer program as a lock-out code. […] [A]n individual who copied a poem solely to use as a password would not have infringed the copyright, because in that scenario, the alleged infringer would have the defense that the poem has 'merged' with a method of operation (the password). By contrast, someone who copied the poem for expressive purposes (for instance, as part of a book of poetry) would not have this defense. For these reasons, I would hold that in cases where the merger is with a method of operation, the merger doctrine should be applied as a defense to infringement only, and not as informing the question of copyrightability of the work itself."[165]

---

[162] Id., pp. 987-989. See also SHUBHA GHOSH, *Legal Code and the Need for a Broader Functionality Doctrine in Copyright*, 50 Journal of the Copyright Society of the U.S.A., 71 (2003), pp. 101—106.

[163] See J. E. TITUS, *Right to Reverse Engineer Software: Is Japan Next and Does It Really Matter?*, 19 North Carolina Journal of International Law and Commercial Regulation, 491 (1994).

[164] The reference is to ROBERT NOZICK, *Anarchy, State, and Utopia*, Basic Books (1974) (the author referred to JOHN LOCKE, *Second Treatise of Government* (1690)).

[165] Lexmark International, Inc. v. Static Control Components, Inc., 387 F.3d 522 at 557—558.

4.3. *Scenes à faire* doctrine

The doctrine of *scenes à faire* – or 'scenes that must be done' – is based on principles which are similar to the ones underlying the merger doctrine, but it is more generous in taking into account external constraints, which are not necessarily immutable over time:

> "when external factors constrain the choice of expressive vehicle, the doctrine of 'scenes a faire' […] precludes copyright protection. In the literary context, the doctrine means that certain phrases that are standard, stock, […] or that necessarily follow from a common theme or setting may not obtain copyright protection. In the computer-software context, the doctrine means that the elements of a program dictated by practical realities – e.g., by hardware standards and mechanical specifications, software standards and compatibility requirements, computer manufacturer design standards, target industry practices, and standard computer programming practices – may not obtain protection."[166]

It must be noted that the *scenes à faire* doctrine, according to some commentators, "lends itself to abuse".[167] Indeed, one has to be very careful not to read this doctrine in such a way as to penalize successful firms just because of their success. The problem is that, if an undertaking created a *de facto* standard and if the author owned some intellectual property rights to it, it would not be fair to tell him or her that this creation is no longer protected, precisely because it was so successful to become an industry-wide standard. In fact, I share the scepticism of part of the literature about the opportunity of embracing such a doctrine, in particular because its effect could be reached through other doctrines and/or general principles. First of all, whenever the control over a *de facto* standard is also coupled with a dominant position in a relevant market (having the competition policy meaning of the expression), the exercise of intellectual property may entail "special responsibilities" under competition law. Moreover, and more generally, I submit that it is possible to deal with the need of accessing compatibility requirements and similar technical issues by just using merger doctrine (or directly adopting the idea/expression dichotomy). About "industry-wide programming practices", what should be determined is simply if these practices entail any protected expression and – if this is the case – if some developers are able to claim to have created them. Otherwise, these programming solutions may be expressive, but they are not original (or –at least – there is no developer who could claim any copyright on them). To be sure, another thing to verify is if it is possible to use a defense of independent creation.

Overall, I do not see much need for a *scenes à faire* doctrine, unless it is interpreted as applying to cases where the idea/expression dichotomy and its corollaries do not apply. A possible example of such an application – but I have to stress that this is just a theoretical suggestion, and not a restatement of the actual application of the doctrine in front of US courts – could concern cases, in which a certain kind of behaviour may prevent a rightholder from enforcing his intellectual property rights. For instance, that may happen if a given rightholder allows other developers free ride on his intellectual property rights in the first place and then changes his attitude, after having established significant market power, thanks to network effects, learning costs and other sources of switching costs. In that way, the creator being "victim of his own success" would – in some way – be penalized, but just in order to protect the reliance of other players in the industry (a reliance that he tolerated or helped in creating). Such an interpretation of the doctrine would bring significant resemblance with analogous doctrines applied in the field of trademarks: some analogies may exist with trademark commoditization or "genericide,"[168] but I am referring, more specifically, to the equitable defenses of estoppel (complex doctrine, barring from claiming or denying an argument on equitable grounds), laches (sometimes called "estoppel by delay", it is related to undue delay in seeking redress), and acquiescence (similar to laches, but involving a more active or conscious conduct).[169] Because these are

---

[166] Lexmark v. SCC (*supra* note 165) at 535—536. Internal citations omitted. Other relevant cases mentioning the doctrine include Chamberlain Group, Inc. v. Skylink Techs., Inc., 381 F.3d 1178 (Fed. Cir. 2004), cert. denied, 544 U.S. 923 (2005); Sony, 464 U.S. 417; Computer Assocs. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992); Sega, 977 F.2d 1510.

[167] See for instance TITUS, *Reverse Engineer Software: Is Japan Next?*: "When applying the scenes a faire doctrine, courts should keep in mind that clichés and conventions are often unoriginal only at the level of their ideas, not their particular expression. The presence of a clichéd scene or section of code may not be dispositive of copying, but this is true only so far as the scene or section is considered in its entirety. The scene may permit a large number of possible expressions, so that excluding plaintiff's particular expression from the substantial similarity analysis is improper."

[168] This well-known doctrine concerning trademarks implies that these marks may become generic nouns or verbs if their owners do not monitor their use and enforce their exclusive rights (resulting in what is sometimes called the "genericide" of the trademark).

[169] For more details about laches and acquiescence, see JAMES LOVE HOPKINS, *The Law of Unfair Trade: Including Trade-Marks, Trade Secrets, and Good-Will*, Hein Publ. (1997) 437, 123 (also available at http://books.google.com/books?id=8JdfuGPHt8UC).

common law equitable defences, their analogical application to copyright could be envisaged.[170] The common point behind these doctrines is that, if one does not enforce one's exclusive right for a sufficiently long period of time and/or in peculiar situations, then one may loose one's right to do so at all. Indeed, there are good reasons to suspect that software houses may experience similar situations. This may be the case when they want to create some reliance on the fact that a set of interfaces – possibly including their implementations[171] – will be a free common pool, from which the industry will be able to tap forever. Indeed, as I already hinted, during the introductory phase of a new software technology firms may actually engage in "evangelization campaigns" with the goal of spreading the adoption of the technology and generating a critical mass of producers of complementary products, able to create sufficient network effects to trigger massive adoption by users. If a firms want to engage in a credible promise to leave its interfaces in a common pool of resources for the industry, the aforementioned doctrines may make this "implicit promise" more credible. To the opposite, the legal system may want to prevent firms from suddenly shifting from an attitude of widespread free (and actively encouraged) use of their technology to one of strict enforcement of their intellectual property rights. Again, the aforementioned doctrine may be useful to achieve this goal.

4.4. Fair use

Fair use is a well-known US common law doctrine, also incorporated into the Copyright Act of 1976. This doctrine is different from the previous ones in as far as it does not concern the scope of copyright protection. Instead it shields from copyright infringement and allows reproducing copyrighted materials by virtually any means, as long as the use of such materials is deemed "fair" and related to some purposes, as "criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research" (but this listing is non-inclusive). According to 17 U.S.C. § 107, four non-exclusive factors should be considered in determining whether a particular use is a fair one: (1) the purpose and character of the use (including whether commercial or for non-profit or educational purposes); (2) the nature of the copyrighted work; (3) the amount and substantiality of the portion used (in relation to the copyrighted work as a whole); (4) the effect of the use upon the potential market for or value of the copyrighted work.

An application of the fair use doctrine will be needed, in order to deal with software interfaces, only as long as the interface under examination is protected *and* the use that is being made of this interface potentially infringes copyright (unless a fair use is found). Hence, in cases in which the idea/expression dichotomy tells us that a certain element is on the "idea side of the divide", it is not necessary to recur to fair use. Additionally, even if a given implementation is protected by copyright, one may read it (assuming one has access to the source code) and write down a specification, which may be abstract enough not to be considered a derivative work of the existing specification. In this case, once again, we do not need to resort to fair use. However, when one needs to perform several copies of the distributed object code of a piece of software in order to decompile and understand it, then a copyright object is copied and – without fair use – the analyzer may be in trouble.

That having been said, notice that, for economists, fair use may be a very tempting doctrine. This is the case, because it has the potential for transforming any rigid rule existing in copyright law into a "rule of reason" analysis, giving significant weight (thanks to the fourth "step" of the test) to the economic impact of the use under examination (with particular focus on the incentives to create). At the same time, there are also sound reasons for not resorting to fair use, unless strictly necessary.

4.4.1. Risks in applying fair use to determine the legal status of interoperability information

Indeed, fair use is so flexible that it may be prohibitively difficult to apply in a predictable way. Hence, it may lead to such a degree of uncertainty that – at the end of the day – recurring too frequently to fair use,

[170] Notice that – at least in the field of trademarks – these doctrines do have equivalents also in civil law, for instance the "Limitation in consequence of acquiescence" established by article 53 of the European Regulation on the Community trade mark. [Council Regulation (EC) No 40/94 of 20 December 1993 on the Community trade mark; Official Journal L 011, 14/01/1994 P. 0001 – 0036.] However, there is no hope of applying similar provisions to copyright by way of analogy, so a specific norm would be needed. If similar and directly applicable norms have to be found in civil law, one should probably look for them in the more flexible field of unfair competition.

[171] Notice that, if a doctrine similar to estoppel is applicable to copyright, its application could be relevant not only for API specifications (in legal systems possibly protecting them), but – more generally – also for standard and publicly available implementations.

which is essentially an *ex post* analysis that is very sensitive to the smallest details of each case, may leave "no use" for "fair use", ultimately harming the status of the doctrine itself. And, since I think that it is healthy for the copyright system to keep fair use as a "doctrine of last resort" – giving the possibility to avoid economically paradoxical effects of formal copyright rules – I suggest avoiding fair use where alternative tools are available.[172] This is why I think that it is lucky that, following the approach proposed in this paper, fair use is not a central doctrine in determining the legal status of software interoperability information. Indeed – and pardon me if I repeat this important point – one has to bring into play fair use just as a tool allowing reverse engineering (moreover, in the second paper of the dissertation, I will actually argue that a clear-cut statutory safe harbour for decompilation would be preferable). In addition, fair use could be used to excuse the (non strictly necessary) literal copying of some lines of API implementations, even in cases where a strict application of merger and *scenes à faire* doctrines fail (and only as long as this copying does not significantly compromise the legitimate interests of the copyright holder).[173] Of course, this is still a quite useful role for fair use, because it may deter useless litigations and reduce transaction costs in general when some copying may have taken place, but without actual prejudice for the original creator. In these cases, fair use could solve any dispute, banning an infringement claim, even if there is some (limited and economically irrelevant) copying of the "expression of an idea" (implementation). However, one should notice that this is just a marginal use of the doctrine, which – in general – is not crucial for the issues at hand in this paper.

One of the main problems of fair use (if used as a general tool to deal with API issues) is that applying the traditional tests (economically analyzed, for instance, by W. J. Gordon)[174] one could mistakenly end up saying that copying is allowed to the benefit of the producers of complementary products, but not of the producers of competing products[175] (see § 7 for a fuller discussion of this issue). As I am going to show, this conclusion would be legally incorrect, but notice that it is also economically inconsistent. Indeed, in these cases – when we deal with complementary products – platform controllers are already inclined to freely disclose API specifications (and maybe also kit to create compatible implementations),[176] so that fair use could only be used in cases in which we do not need it.

To summarize, I argue that fair use should not be considered as the main legal tool in solving the general problem of API protection. That is true because I submit that API specifications should not be protected in the first place, in particular using the merger doctrine or directly applying the idea/expression dichotomy. Moreover, this choice will also turn out to be useful in adapting to the EU the conclusions that one may draw analysing US case law. That is the case because fair use is a strictly common law (and especially American) doctrine, while it is easier to argue that the general idea/expression dichotomy (and the merger and – possibly – *scenes à faire* doctrines as special qualification of this general distinction) are essentially common to the US

---

[172] For a fuller law & economics analysis of the problems of fair use (even though that analysis is much broader and hence stresses much more the usefulness of fair use), see THOMAS F. COTTER, *Fair Use And Copyright Overenforcement*, 93 Iowa Law Review, 1271 (2008). The author is also proposing several measures to mitigate some of the weaknesses of fair use, including its "notorious unpredictability" (see p. 1312).

[173] If literal copying is necessary for technical reasons, either there is no copyright protection at all or the merger doctrine applies, hence fair use is not necessary. Actually, there is at least another case, in which fair use may seem to be necessary. Think, in particular, of a case in which an original poem is used as key to lock a system. In that case, a formal violation of copyright would be needed in order to obtain a functional result. However, in that case, also the merger doctrine would apply and – in my opinion – the optimal solution would be the one of recurring to copyright misuse (see , § *9.1.1. Licenses and copyright misuse*). Hence, it is almost only in case of reverse engineering that – given the peculiarities of software – fair use is strictly necessary, but this is a topic that will be discussed in the second paper of this dissertation.

[174] WENDY J. GORDON, *Fair use as market failure: A structural and economic analysis of the Betamax case and its predecessors*, 82 Columbia Law Review, 1600 (1982). For a recent work and more references, see COTTER, *Fair Use And Copyright Overenforcement*. My idea of not proposing fair use as a general solution is also related to the fact that the main economic rational for fair use remains linked to transaction costs, but in the field of software we are not dealing with a lot of users wanting to quote some lines of code of the incumbent; we are dealing with some other developers wanting to produce software able to interoperate with that of the incumbent. It would be (it actually is) easy to reach an agreement between the producers of complementary product and the incumbent itself to obtain a license (or the disclosure of a trade secret, with an agreement for non-disclosure to third parties). In such a setting, a more compelling explanation of fair use could be related not to transaction costs, but the the risk of tragedy of the anti-commons. *See* BEN DEPOORTER & FRANCESCO PARISI, *Fair use and copyright protection: a price theory explanation*, 21 International Review of Law and Economics, 453–-473 (2002). However, I will show that we may solve this anti-commoms problem, without recurring to fair use and hence increasing legal certainty and avoid the side-effects of a complex fair use analysis.

[175] This result is also similar to the solution proposed by WEISER, *The Internet, Innovation, and IP Policy*.

[176] All that, of course, would be coupled with non-disclosure agreements to avoid spillovers in favour of competing platform producers.

and EU legal systems, at least in the field of software.[177] Moreover, since I already anticipated my conclusion about the fact that vertical and horizontal interoperability should be treated in the same way, let me also anticipate a *caveat*. I will suggest that the distinction between producers of complements and competitors should normally matter in a later stage, for unfair competition or antitrust analysis (again, leaving the details about antitrust for the third paper).

4.4.2. Decomposing the access phase and the re-implementation phase

The fair use analysis may be misleading, in particular if we do not carefully avoid blurring the distinction between the direct product of decompilation and the use of the information obtained. In fact, I submit that fair use analysis should concern (only, or – at least – primarily) the activity of accessing the original API implementation, fairly copied, as a step in reconstructing a proxy of the original specification. In this context, what is lost to the right holder is (at most) the income associated with the direct licensing of a copy of the source code of the original implementation (which is not the "normal exploitation of the work"[178]). Moreover, the substitute generated by decompilation is a very imperfect one and probably not a very good substitute, if the original developer has any intention to license (the access to) the original implementation. Indeed, decompilation is a very costly process (as I will show in the second paper) and it is carried out if and almost only if there is no way to "buy" a copy of the original source code at a lower price.[179] In this context, the only prejudice to the right owner generated by reverse engineering is a price cap on his possibility of exploiting his market power by licensing the original specification (a cap which is set at a high level, typically higher than the entire initial development cost of actually writing down the code). Alternatively – in case the right holder decides not to license at all the original specification – decompilation generates just a competitive threat, coming from a new, independently created program, implementing a reconstructed and imperfect proxy of this specification. It is difficult to depict either of these "prejudices" (if any) as an "unreasonably prejudice the legitimate interests of the right holder".

Once again, what we should not do is apply the fair use test to later uses of the reconstructed specification, which is not a "derivative work" of the reconstructed implementation, at least no more than the solution of a problem using a theorem learned in a maths book is a derivative work of the original maths

---

[177] As it has been shown by DERCLAYE, *Software Copyright Protection -- Part 1* and DERCLAYE, *Software Copyright Protection -- Part 2*. Notice also that article 1, par. 2, of the Software Directive recalls that: "Protection in accordance with this Directive shall apply to the expression in any form of a computer program. Ideas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright under this Directive."

[178] Article 13 of TRIPS obliges member states to "confine limitations or exceptions to exclusive rights to certain special cases which do not conflict with the normal exploitation of the work and do not unreasonably prejudice the legitimate interests of the right holder". In fact, also the fourth step of the fair use test can be interpreted as concerning mainly the "normal exploitation of the work". About this point, one should consider the approach recalled by ANDREA OTTOLIA & DAN WIELSCH, *Mapping the Information Environment: Legal Aspects of Modularization and Digitalization*, 6 Yale Journal of Law and Technology, 174 (2004), text accompanying footnotes 349-355. In *American Geophysical Union v. Texaco, Inc.*, the Second Circuit "stated that, in order to be relevant under the fourth factor, the character of the potential market had to be either 'traditional, reasonable, or likely.'" A possible criticism to this approach – along with a convincing response – is discussed by Ottolia and Wielsch, text accompanying f.n. 371. "The main criticism raised against the Texaco approach has been one of circularity. If the core of the fair use assessment is to verify the fairness of a use and the consequent impossibility of a copyright owner controlling and licensing such a use, then the existence of a potential market might be the result in the negation of fair use rather than its justification. The circularity criticism is valid where the adopted concept of fair use is based on the assumption of a continuing balance between freedom and control as existing in the functional constitutional propertarian model. If the existence of propriety over information is only justified by the need to stimulate creativity, the ability of fair uses to foster such innovation must also be recognized. However, if the reasoning is based on a market failure model, then it is not circular. If the ultimate justification of fair use is a failure in the market mechanism for a certain use, the existence of a market for such a use, undertaken without the copyright owner's consent, impedes its qualification as fair use. The court in Texaco seems to retain such a market failure approach. The limits put forth on such an approach – 'traditionality,' 'reasonableness,' and 'likelihood' – are not inconsistent with it."

[179] As noticed (among others) by ROTENBERG, *Regulation of Software Interoperability*, decompilation is just increasing incentives to license this access at a low price: "By allowing access to a standard, intellectual property law creates an incentive for the standard bearer to licence its product to rivals at an amount equal to the cost of reverse engineering the platform standard. […] In fact, the price will normally be a little higher than that because the platform owner has valuable specification information above and beyond access to the technical interface itself." And – what is very important – the cost of a decompilation project that reconstructs a good proxy of the original source code (that becomes a proxy of the maximum price to license access to the original source code – being the opportunity cost of accessing it) may actually be higher that the cost of writing the original code from scratch: the fact that someone may be willing to pay more that the cost of writing the code from scratch just to study it derives from network effects, learning costs and other "added values" that producers of complementary products and users generated "around" the original software.

book. This approach is coherent with what has been proposed (already during the '80s in the US) by supporters of the legality of software reverse engineering, as reported by McManis,[180] who is precisely stressing that

> "[t]he fairness of reverse engineering should not be confused with the fairness of the ultimate uses made of the product of the reverse engineering or with the potential effects of such uses on the market for the publicly distributed program." [And this precisely because doing so would fail to appropriately take into account the idea/expression dichotomy], i.e. "[fail] to distinguish between creating a competing program that uses copyrighted expression contained in the copyrighted program, and creating a competing program that merely accomplishes the same function or uses unprotected ideas found in the copyrighted program. While the former situation would raise a legitimate fair use issue, the latter would not."[181]

To avoid any confusion, I suggest one to imagine that two different firms are performing the access phase (needing reverse engineering and the creation of several derivative works of the original interface implementation, as intermediary steps to reconstruct technical principles at the core of the original specification) and the reimplementation phase (which may be based on the final product of the access phase, that may – or should – be a simple reconstructed proxy of the original interface specification). Alternatively, one may imagine that a single firm is decomposing the construction of an interoperable process adopting the clean-room/dirty-room process discussed above.

The access phase surely involves formal violations of copyright. Starting from a (standard) licensed copy of the original object code (available in commerce for end users), several intermediate copies of this code are created, leading to a derivative work – the reconstructed proxy of the original source code – which is likely to be creative in itself, but which is also a derivative work of the original protected expression. This is why I think that a direct commercial use of the code reconstructed during the access phase would be a blatant violation of the original copyright and should not enjoy a fair use exception (having being created to be as close as possible a substitute of the original work, both in a functional and in an expressive sense). However, the access phase does not typically involve any direct commercial use of its product. Instead, the reconstructed code is used "privately" and for purposes which are of study (and possibly of criticism and/or teaching). Hence, the true final product of the access phase may not be the proxy of the original source code, but a technical manual describing it, in other words a "manual of instructions" describing how to write another software able to interoperate with the decompiled one (or with third parties' software compatible with the decompiled one). *Id est*, an "interface specification manual". To avoid a quasi-equivalence between copyright law and patent law in the field of software, I argue that it must be possible to write a version of this manual, which is not in violation of the original copyrighted work. As in the case of a new instruction manual for a given hardware (think about a TV set), even if the new "manual" is written without looking at the original one (but just studying the hardware itself), there will be apparently relevant similarities between the two documents, but if we filtrate[182] needed technical parts ("push power button to turn the TV on") these similarities are likely to disappear (or significantly fade away). The classical fair use test may be used to understand if the "interface specification manual" created (i.e. the only direct output of decompilation) is or not in violation of the copyright on the original work. But it is important to notice that this specification manual: (1-2) will have the purpose of accessing technical ideas, process and methods; (3) will (should) copy only what is needed in order to technically achieve this purpose or lines of code which it would be technically awkward to write otherwise; (4) will not be in direct competition with the original implementation, apart from the specific and narrow market of developers wanting to license access to the original source code for interoperability purposes – and, also in this market, only as a very imperfect substitute and only if it is

---

[180] C. R. MCMANIS, *Intellectual Property Protection and Reverse Engineering of Computer Programs in the United States and the European Community*, 8 Berkeley Technology Law Journal, 25 (1993), quoting the *LaST Frontier Conference Report on Copyright Protection of Computer Software*, 30 JURIMETRICS J. 15 (1989) in particular at 24-25.

[181] Id.. In fact, the fair user "may make significant use of prior work, so long as he does not bodily appropriate the expression of another". *See* Hoehling v. Universal City Studios, 618 F.2d 972, 980 (2d Cir. 1980), quoted by MCMANIS, *IP Protection and Reverse Engineering*. The author also argues that "Professor Miller appears to make the converse mistake when he states that one reason for prohibiting decompilation and disassembly is '[b]ecause the traces of copying can be disguised.'" (see A. R. MILLER, *Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU?*, 106 Harvard Law Review, 977 (1993) at 1027.) Once again, "[t]he fairness of the initial copying (i.e., decompilation or disassembly), and the fairness of the subsequent use of the product of the reverse engineering, must each be determined on its own merits."

[182] The verb "filtrate" is used in the sense of the Altai (or abstraction-filtration-comparison) test, discussed above (see § *4.1.2. Adoption of the "three-step test" of Computer Associates v. Altai*).

available to third parties (otherwise it will just prevent the sale of a single license of this kind to the decompiling firm, but will not influence the rest of this peculiar market for interoperability).

At this point we may shift to the reimplementation phase, where a completely new software code is written from scratch, but respecting the indications of the "manual", which may mandate – in certain cases (hopefully only if technically needed) – some lines of code or specific expressions. If writing the "manual" was deemed a fair use – and if the manual mandated the only reproductions of "expression" that took place – I argue (see also section 7) that this reimplementation phase cannot violate copyright. The fair use analysis should be focused just on the first phase.

Before moving on to the European legislative framework, let me stress again that the two "steps" I described (i.e. access and reimplementation) may actually be not so neatly divided. In practice – unless a firm actually followed a clean room process – it is possible that there is no "interface specification manual" at all. However, courts will always have the possibility of confronting (as a mental experiment and helped by experts testifying for the two parties) the actual reimplementation with the one that would have been written following a hypothetical non-infringing "interface specification document".

5. Investigating the legal status of interoperability information: the European setting

In 1989, in a short but very insightful paper,[183] W. R. Cornish stressed that the legal status of interoperability information was a relevant issue that deserved a clear solution in the proposed *Software Directive*, which was being discussed at the time. In particular, the author appreciated what the Commission already stressed in the draft directive at article 1(3):

> "protection ... shall apply to the expression in any form of a computer program but shall not extend to the ideas, principles, logic, algorithms or programming languages underlying the program".[184]

However, according to the author, that "unambiguous statement [was] at once clouded by a further sentence", specifying that "[w]here the *specification of interfaces* constitutes ideas and principles which underlie the program, those ideas and principles are not copyrightable subject matter". The problem was that, according to Cornish, "*[s]pecification of interfaces, as distinct from their implementation in code*, necessarily constitutes 'ideas and principles'." Instead, the wording of the Commission introduced the suggestion that this may not always be the case. According to the author, that wording could have created "a puzzle which can only be resolved by courts in expensive litigation."[185] Despite the fact that the final wording of the Software Directive was different from the one commented by Cornish, we may say that he was right, indeed, since the puzzle he refers to is still to be solved.

Actually, as I already showed, there is a way to reconcile the opinion of Cornish and the aforementioned wording of the Commission. Abstract interface specifications are always unprotected ideas, principles and methods, as Cornish argued. However, a given specification document may contain protected expression and it is normally protected by copyright, at least against straightforward literal reproduction, as the Commission implied. In any case, the final version of the *Software Directive* eliminates the specific ambiguous wording criticized by Cornish. In fact, article 1 ("Object of protection") recites at par. 2:

> "Protection in accordance with this Directive shall apply to the expression in any form of a computer program. Ideas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright under this Directive."

Notice that the Directive speaks about "interfaces" in general, not "interface specifications", hence it is possible to read it as suggested by Cornish and as recommended in the paper at hand: implementations are protected, but not specifications (with or without accepting the additional qualification I proposed concerning abstract interface specifications and specification documents). Moreover, point 13 of the *Software Directive*'s Preamble also clarifies that this principle is a general one of copyright law, and it is stated just "for the avoidance of doubt". In other words, the Preamble recognizes the idea/expression dichotomy as an undisputed fundamental principle of copyright law also in the Europe Union.

However, some ambiguity remains in the final text of the Directive; actually, ambiguity is probably increased. Indeed, the final text is perfectly compatible with the thesis of this paper or with the thesis of

[183] CORNISH, *Inter-operable Systems*, . Among other things, this paper urged the Commission to introduce a specific decompilation exception (see page 392).

[184] See Id., 391.

[185] *Ibidem*. Emphasis added.

Cornish, but it is also vague enough to be compatible with other readings. In particular, the word "specification" completely disappeared from the approved Directive, so that it is still technically possible to argue that there are cases in which interface specifications are somehow copyright protected (possibly also abstract interface specifications). The choice of leaving some ambiguity in the final text of the Directive may derive from the fact that no clear answer can be given about the legal status of "software interfaces" in general. However, it is probably legitimate to suspect that the equilibristic exercise of political balancing performed by the Commission required some ambiguity to remain, and prevented the introduction of a clearer statement saying that "the expression of interface implementations may be protected by copyright, according to the general principles of copyright law, but ideas, principles and methods underlying interface specifications cannot be protected, so that the realization of an original interoperable interface implementation shall always be allowed." In any case, and despite the absence of such a clear statement, I argue that the *Software Directive* has to be read in this way and, to show that, I will start from the analysis of its article 6.

The first paragraph of article 6 of the *Software Directive* explicitly addresses software reverse engineering ("decompilation"), authorizing reproductions of copyrighted programs "where reproduction of the code and translation of its form […] are indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs". The same article also provides several conditions to be met, in order to enjoy the decompilation "privilege": (1) the initial copy of the program to decompile must be legally owned by the reverse engineer; (2) the information necessary to achieve interoperability must not be already "readily available"[186]; and (3) reverse engineering is "confined to the parts of the original program which are necessary to achieve interoperability". These conditions are very similar to the ones required by US courts to find a fair use exception for reverse engineering (at least if they are interpreted in the same broad way in which they are in the US).

Also the second paragraph of article 6 is interesting (and complex) to analyze (and eventually criticize, as I will do in the second paper of this dissertation). It states that:

> "[t]he provisions of paragraph 1 shall not permit the information obtained through its application: (a) to be used for goals other than to achieve the interoperability of the independently created computer program; (b) to be given to others, except when necessary for the interoperability of the independently created computer program; or (c) to be used for the development, production or marketing of a computer program substantially similar in its expression, or for any other act which infringes copyright."

These condition could be interpreted in a "very narrow way", saying that: (c) an original re-implementation of any interoperability specification cannot infringe copyright, unless it reproduces technically unnecessary expression (economically, you cannot free-ride on the sunk, up-front cost of writing code lines, but you are free to re-implement underlying ideas); however, (b) you are not free to sell "derivative works", generated from the original program through reverse engineering; and, to be sure, (a) you cannot devise a way to violate copyright thanks to this exception. However, and despite its incoherence with general copyright principles, condition (b) also seems to imply that "raw information" and more abstract ideas and methods (like API specifications!) are forced to remain "confidential" if obtained thanks to the decompilation exception of article 6. In my opinion, this would be an excessive reinforcement of trade secret – in the direction of a quasi-property right, as found by Ghidini-Falce for the recent Italian legislation[187] – and an unreasonable disadvantage for decentralized and collaborative models of development, like the open source one.[188] Yet, I will not discuss further this specific issue here, since I deal with it in the second paper of this dissertation. For the moment, I will simply interpret article 6.2 following Samuelson and Sctotchmer:[189]

---

[186] According to several commentators (see, for instance, Johnsons-Laird, footnote), interoperability specifications for software are dramatically complex: differences between the theoretical specification and actual requirements for "its" implementation may not be completely clear, not even to the original developer. It may be naive to think that the fact that some interoperability information is available and it is supposed (even *bona fide*) to be sufficient could ban reverse engineering and further investigation.

[187] See General Introduction and Gustavo Ghidini and Valeria Falce, "*Recent developments in Italian regulation of trade and industrial secrets: A patent contradiction of the patent regime?*", preliminary draft presented at the 3rd Annual Workshop on The Law and Economics of Intellectual Property and Information Technology, Centre for Commercial Law Studies, Queen Mary, University of London, 5-6 July 2007.

[188] Open source projects, because of their inherent tendency to spread information, may present higher risks of potentially harmful distribution of (copyright protected) intermediate copies. But focusing on specific problems of open source decompilation projects is not just a curiosity, since the most credible projects to replicate windows APIs are being conducted by open source programmers.

[189] SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*.

> "[t]he EU rule essentially requires each firm that wants to reverse-engineer to bear the full expense of decompiling the program on its own. This preserves the lead time of the firm whose program has been decompiled, but leads to more socially wasteful costs unless the software developer licenses interface information to foreclose the decompilation effort."[190]

Also paragraph three (6.3) is economically interesting:

> "In accordance with the provisions of the Berne Convention for the protection of Literary and Artistic Works, the provisions of this Article may not be interpreted in such a way as to allow its application to be used in a manner which unreasonably prejudices the right holder's legitimate interests or conflicts with a normal exploitation of the computer program."

This is clearly a common principle to almost any jurisdiction, but – for instance – US courts interpreted similar arguments allowing not only "vertical interoperability"[191] (that is, interoperability with complementary goods, like another application for a given operating system), but also "horizontal interoperability"[192] (allowing, for instance, decompilation to be used to realize a competitive operating system, which is able to run the same applications as the first one, maybe running on a different hardware).[193] Again, this is an issue more concerning reverse engineering than the legal status of interoperability information – so I will discuss it later – but I can anticipate that the Directive can be read as allowing the achievement of both vertical and horizontal interoperability, as it is confirmed by the Preamble, where recital 21 clearly talks about "reproduction of the code and translation of its form […] indispensable to obtain the necessary information to achieve the interoperability of an independently created program *with other programs* [plural]" and *not just with the decompiled program*. Despite the fact that some commentators interpreted the very same legal text, suggesting that article 6 actually allows only vertical interoperability,[194] the reading I propose is strongly backed by the general reading of the Directive, proposed by the Commission (and confirmed by the Court of First Instance) in the recent Microsoft Case.[195]

Another point deserves some attention. Article 6 is barring copyright (holders) from using copyright law to make it impossible to access interoperability information through reverse engineering. However, if interoperability information is achieved thanks to the decompilation exception, there seem to be constraints to its disclosure (constraints that do not bind, in case the same information is otherwise obtained, for instance through other reverse engineering techniques – like the black box analysis – not involving decompilation). Hence, as confirmed by the interpretation of Samuelson and Scotchmer,[196]

> [o]ne cannot, for example, publish information learned during reverse engineering [*rectius*, decompilation]. This puts at risk authors of books such as Andrew Schulman et Al., Undocumented Windows: A Programmer's Guide to Reserved Microsoft Windows API Functions (1992). Under article 6(2), European decompilers are at risk if they try to recoup their reverse engineering expenses by licensing the information they learn in the course of their reverse engineering efforts. The official commentary to the European *Software Directive* asserts that article 6(2)(b) 'prevents the publication or trafficking in information by those who have decompiled existing programs, since it would be inequitable to impose conditions on the decompiler but allow others access to the information which he had then made public.'

As the second paper of the dissertation will show, the limits imposed on the disclosure of the "information" obtained through decompilation are economically unnecessary and inappropriate. In fact, it would have been

---

[190] In fact, the authors already argued that a possible "policy option is to allow reverse engineering, but to forbid publication or other disclosures of information obtained thereby. For the most part, the law has not had to address this issue because reverse engineers have generally had little incentive to publish or otherwise disclose information they learn from reverse engineering. Reverse engineers have typically kept the resulting know-how secret for competitive advantage." But this is not the case for open source software projects.

[191] Vertical interoperability may also be called direct interoperability. See § *7. Vertical and horizontal access; transformative and substitutive uses*.

[192] Horizontal interoperability may also be called indirect interoperability. See § *7. Vertical and horizontal access; transformative and substitutive uses*.

[193] I added this specification, because the US decision I am referring to (Sony v. Connectix, 203 F.3d 596) concerned a case in which "horizontal interoperability" allowed to play the same applications (games) on a different hardware: it is not clear, from the case, if this fact has been determinant; it surely did weight in the direction of fair use (and – to use the language of the Directive – against the finding of an "unreasonable prejudices to the right holder's legitimate interests").

[194] See, in particular, R. J. HART, *Interoperability Information and the Microsoft Decision*, 28 European Intellectual Property Review, 361--365 (2006).

[195] See *Microsoft CFI*, § 118 ff.. See also the third paper of this dissertation project for further details.

[196] SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*.

much more sensible to impose these constraints just on derivative works of the original implementation (which are copyright protected expression and represent the intermediate products of decompilation), leaving at the same time complete freedom to disclose the *indirectly* obtained interoperability specification (which is the "information" to which one really wanted to "access" and that it is not protected by copyright, according to the view expressed in this paper). That having been said, these constraints are binding law.

Overall, I share the view of part of the literature,[197] according to which the European legal setting is fairly clear, at least in the field of copyright. Carefully considering article 6, one has to conclude that "it became well established that interoperability information can't be generally copyrighted. Copyright can only apply to e.g. specification documents but it can't prohibit in any way independent implementations of the specifications themselves."[198] In fact, even if article 6 is (unfortunately) hardly as clear as some authors seem to imply, from a systematic reading of the Directive it is undisputable that the discussed article would be meaningless if it was not legitimate to use (in independently recreated implementations) the information obtained from decompilation. So, the absence of a specific exception to *use* interoperability information could be read in two ways. Either there is an implicit (but quite obviously existent) exception allowing to do so (despite the existence of copyright protection on interface specifications). Or – much more likely – the legislator considered as obvious the fact that these pieces of information (or ideas?) where not protected at all. I favour the second explanation, also because copyright exception and limitation are typically expressly stated in civil law countries (and in European countries in general), and not doing so would have been at least imprudent at the European level.[199] In any case, we must conclude that using the obtained interoperability information (obtained following the procedure detailed in Art. 6) is legitimate in Europe.

5.1. European doctrines allowing literal copying

Article 6 is an exception allowing the reproduction of software code without the authorization of the rightholder for a specific purpose; however, this exception does not extent to the further use of the implementation reconstructed through decompilation. This is not a major problem, since – as I already mentioned – literal copying is not necessary (in principle) to re-implement a specification, unless there is a merge of ideas and expression, and in this case the expression is not protected by copyright in the first place. At the same time, I mentioned that, in the US, the fair use doctrine could allow also some literal copying, as long as the four-step test required by the Copyright Act weighs in favour of the user. However, the fair use doctrine does not apply to Europe, hence it may be interesting to verify if there are other exceptions or limitations allowing some limited literal copying outside the US. A starting point to check this possibility may be the "quotation right," provided by virtually any civil law jurisdiction. Unfortunately, according to the majority of authors,[200] a direct application of similar exceptions to software interfaces is at least disputable in several countries, including Germany and the Netherlands (and, I may add, Italy).[201] Other jurisdictions, as France, may possibly leave some more room for broader interpretation of the quotation right:

> article L. 122-5 of the French Act seems to be worded somewhat more broadly, making quotation permissible provided that mention is made of author and source, and the use involves 'brief quotations which are justified by – inter alia – the scientific or informational character of the work into which they are incorporated.' Whether this will be sufficient to cover interfaces remains to be seen."[202]

---

[197] See, in particular, VÄLIMÄKI, *Software Interoperability and IP*.

[198] Id., 4.

[199] See, for instance, ANNE LEPAGE, *Overview of Exceptions and Limitations to Copyright in the Digital Environment*, January - March UNESCO e-Copyright Bulletin, 1--19 (2003), § *Closed systems of exceptions* (pp. 6 ff.).

[200] See, for instance, SPOOR, *Copyright Protection and Reverse Engineering*.

[201] See SPOOR, *Copyright Protection and Reverse Engineering*, 1075—1076: "According to Bauer [indirect quotation of K. BAUER, *Reverse Engineering und Urheberrecht*, 6 Computer und Recht 89 (1990)], article 51 of the German Copyright Act, which deals with the right of quotation, does not directly apply to interfaces, nor indeed does any other existing exception. In his view an extensive interpretation might perhaps be justifiable, but it is unlikely to be accepted by the German Supreme Court, given that court's usual narrow interpretation of such exceptions. Dreier, however, does not consider the application of the quotation right to be excluded, although he believes it must remain restricted to 'the extent necessary for the purpose.'" […] "The bill to amend the Dutch Copyright Act originally provided that none of the ordinary exceptions to copyright would apply to software. This provision, however, was limited by Parliament to the effect that only the exception which allows the copying of a work for private use shall not apply to software. Although the right of quotation will continue to apply to software, article 15a is worded in such a way that it can hardly apply to the copying of interfaces into other software.

[202] Id., 1075—1076.

Overall, the use of the quotation right in the field of software seems to be quite stretched and probably inappropriate. Apart from the difficulties connected with respecting the formal requirement of the quotation right (e.g. mentioning the author and other details concerning the source of the quotation), a quotation would indeed require some "inherency" of the object of the quotation, meaning that the quoted part must be the object of the discussion. Hence, one may conclude that it is advisable that developers avoid technically non strictly necessary reproductions of protected expression, since courts (and their experts) could consider them as evidence of literal copying and there is no clear exception allowing them in an indisputable way in Europe. This practice should be followed also in the US, as a matter of prudence.

That having been said, when a literal reproduction is found (and/or if a reproduction had been mistakenly performed, thinking that it was technically necessary) in principle it could be possible to directly invoke the article 10, paragraph 1, of the Berne Convention (and/or its national implementations) as a shield:

> "it shall be permissible to make quotations from a work which has already been made lawfully available to the public, provided that their making is compatible with fair practice, and that their extent does not exceed that justified by the purpose."

In fact, there are at least some authors,[203] according to which this article is directly applicable to software copyright issues and all EU countries are member of the Berne Convention.

5.2. According to the commission, several APIs are not innovative in themselves

When technologists[204] try to explain intuitively the role of APIs, they frequently analogize these pieces of software to the "*gear teeth, levers, pulleys, and belts that physical machines use to interoperate*". I am aware that analogies could be misleading, but let me continue with this one: gear teeth, levers, pulleys and belts may be innovative in themselves, but it is much more frequent to find innovative engines and other machines, which are connected to other machines using fairly obvious re-implementations of more or less typical gear teeth, levers, pulleys and belts. Apparently, according to the Commission (as I will try to show in the following paragraphs), the same thing is frequently true for APIs.

Taking into account the advice of the Microsoft Monitoring Trustee (established after the Commission's Decision of March 2004[205]) and of the Commission's technical advisors, TAEUS, Competition Commissioner Neelie Kroes stated[206] that: "*The Commission's current view is that there is no significant innovation in these protocols*" [meaning Microsoft client/server communication protocols]. I suggest that here "innovation", "novelty" and similar concepts are used in their patent law sense: these protocols are actually new – or, at least, not fully copied – and (at least in part) original (in the copyright law sense of the word, i.e. independently written without copying).[207] However, there is nothing in these protocols that could not have been suggested by an average software developer. In other words, these protocols are the typical software object: there is nothing dramatically new in them, but ten developers would likely adopt ten similar (and still different and technically incompatible)[208] solutions. Hence, the Commission is not saying that Microsoft's implementation of these protocols is not copyright protected. Instead, the Commission is implicitly arguing that Microsoft's protocols are not patentable (not in the EU, nor in the US if the Commission's technical advisors are right). The Trustee and the TAEUS also discovered some discrepancies between the innovative content of Microsoft's

---

[203] See in particular Id., 1075-1076. "At least that article does not leave the matter to national law, as do many other Berne Convention provisions concerning copyright restrictions. As Ricketson points out, 'this is a mandatory requirement of the Convention to which each Union member must give effect in relation to works claiming protection under the Convention.'"

[204] The following quotation is taken from SAMUELSON, et al., *A Manifesto*. In this paper, two law scholars and two technologists joined their effort to propose a *sui generis* approach to legally protect software innovation (and – what is more important for my present purposes – they described the patterns of innovation in software markets, the obstacles to subsequent innovation and interoperability and so on).

[205] Commission Decision of 24 March 2004, Case T-201/04 (for a synthesis of this Decision, see IP/04/38 and MEMO/04/70).

[206] Official press release IP/07/269 of 01/03/2007: "Competition: Commission warns Microsoft of further penalties over unreasonable pricing as interoperability information lacks significant innovation"

[207] It is precisely because they are new and original, at least in the copyright law sense of these words (that is, written by Microsoft in a way requiring some arbitrary or discretionary choices), that Microsoft competitors have difficulties in reconstructing them and ask the software house to release them.

[208] The likelihood of incompatibility is even higher in case creating a protocol that is not completely standard in its way to operate is one of the goals of the developer, and this was likely the case for some of the protocols concerned by the decision.

protocols and Microsoft's pricing strategies:[209] why did Microsoft price more some protocols (labelled as "Gold" and "Silver") which seem to be less innovative than the cheaper "Bronze" protocols? The rationale behind these choices is easy to understand: prices do not come just from technical complexity, but also from the value of a protocol in strategic terms (as a trade secret and a tool to manage the market and control interoperability). In particular when we deal with platform controllers acting strategically, there are no reasons to assume that API specifications which require higher investments will be more highly priced than more or less trivial modifications of already well known specifications. Unless regulators intervene, prices will not reflect costs, but the strategic value of each protocol and, in particular, the complementarities (or the risk of potential competition) between pieces of software using this protocol and the ones controlled by the platform leader.

To conclude, and assuming that the technical arguments quoted by the Commission are correct, there is no reason to concede intellectual property rights to Microsoft API specifications, which are just particular versions of other specifications, likely realized with the main goal (or – at least – the main effect) of being specific for Windows and allowing Microsoft to decide which software developer should gain full interoperability and which others should be "left a little bit behind". At the same time, Microsoft's implementations of these (more or less standard and only slightly modified) specifications are likely to be original code, protected by copyright and by trade secret law. And, obviously, intellectual property law is not preventing Microsoft from making a strategic use of trade secret (even if, to be sure, this may be an interesting issue for competition law, that I will address in the third paper of this dissertation).

6. A Japanese perspective

About the protection of software interfaces in general in Japan, the literature seems to share the usual uncertainty:

> [T]he differences among the numerous types of 'interfaces' preclude a simplistic and overgeneralised conclusion either that computer 'interfaces' are or are not protected under the JCL [Copyright Law of Japan].[210] [So that,] Japanese courts will take a case-by-case approach to the issue of copyrightability of computer 'interface' characteristics based on well-established principles of the JCL.[211]

However (or maybe precisely because the kind of "uncertainty" that we face is the same that is frequently perceived in the US or in the EU) there are several reasons to believe that the general analysis performed in this paper applies also to Japan.[212] In fact, the idea/expression dichotomy is an accepted principle of copyright law in Japan, and article 10(3) of the Copyright Law of Japan[213] even contains a specific interpretation of the dichotomy, as applied to software programs:

> The protection granted by this Law to works mentioned in paragraph (1), item (ix) [i. e. to software programs] shall not extend to any programming language, rule or algorithm used for making such works. In this case, the following terms shall have meaning hereby assigned to them respectively:
> (i) 'programming language' means letters and other symbols as well as their systems for use as means of expressing a program;

---

[209] "Microsoft divided the protocols into Gold, Silver and Bronze price categories based on the claimed degree of innovation. Microsoft has already agreed that there is a fourth category of protocols, not necessarily innovative, for which there will be no royalty." […] "The Trustee considers that of the total of 160 claims, only four, relating to relatively minor Bronze protocols, represent even a limited degree of innovation. As regards each of the other claims, the Trustee advised that: 'all of the described features were considered either to have been Microsoft implementations of prior developments by others, or to have been anticipated by prior developments and to be immediately obvious minor extensions to that prior work.' TAEUS, which examined the main Gold and Silver protocols, reached the same conclusions as to lack of innovation." See official press release IP/07/269 of 01/03/2007.

[210] KENICHI NAKANO & OSAMU HIRAKAWA, *Copyright Protection of Computer 'Interfaces' in Japan*, 12 European Intellectual Property Review, 46--57 (1990), p. 47.

[211] Id., p. 57.

[212] Consistently with the general principle of copyright law, as confirmed by Id., , p. 48: "[T]he touchstone of the court's analysis in each case, consistent with the principles of the JCL applied in other contexts, will simply be whether 'expression' (as that term has been defined and developed in Japanese case law) is manifested in the program code (or other form of the interface) in a 'creative' manner. A review of existing decisions of Japanese courts in computer software cases suggests that 'expression' will indeed be found in the program code implementing relevant 'interfaces'."

[213] An English translation of the Japanese Copyright Act is available from the website of the Cabinet Secretariat of Japan: http://www.cas.go.jp/jp/seisaku/hourei/data2.html ("Translations of laws and regulations in compliance with Standard Bilingual Dictionary"; last visited July 23, 2008).

(ii) 'rule' means a special rule on how to use in a particular program a programming language mentioned in the preceding item;
(iii) 'algorithm' means methods of combining, in a program, instructions given to a computer.[214]

Hence, one may conclude that:

> an 'interface' that is no more than a programming language, a rule or an algorithm will not be protected; however, an 'interface' which comprises a unique expression of the programmer, even where it is written in a programming language or incorporates numerous rules and algorithms, will be protected by the JCL.[215]

Moreover, an essentially equivalent conclusion may be rephrased in a less ambiguous way using the language of the paper at hand. So, one may say that an interface specification – precisely stating what kind of grammar and syntax should be used in order to implement a communication among pieces of software or computers – is not protected, while the software code implementing these rules (i.e. an expressed "discourse" respecting the given "grammar and syntax") is protected. (Obviously, in Japan, as in Europe and as in the US, a specific interface specification document – as a book explaining English grammar and syntax – is copyright protected.) Further qualifications risk being misleading.[216] In fact, the specification/implementation dichotomy that I described for APIs and Communication Protocols almost exactly applies to programming languages as well:[217]

> A programming language is an artificial language that can be used to control the behavior of a machine, particularly a computer. Programming languages are defined by syntactic and semantic rules which describe their structure and meaning respectively. Many programming languages have some form of written specification of their syntax and semantics; some are defined only by an official implementation.[218]

Hence, the fact that the Japanese law explicitly excludes the protectability of programming languages (quite unmistakably referring to the abstract general "specification of their syntax and semantics") is another, quite direct, confirmation of the fact that API specifications are not copyright protected in Japan.[219]

7. Vertical and horizontal access; transformative and substitutive uses

Both in discussing the appropriate interpretation of interoperability in the *Software Directive* and in analysing the US fair uses doctrine, I already incidentally touched upon a question: should we distinguish (for the purpose of determining the legal status of software interoperability information) between completely transformative and alternative uses on the one hand and uses which are transformative, but lead also to the creation of substitutive products on the other hand? In other words, should we distinguish between uses that need the original product as an inspiration to understand technical principles, methods and ideas, but then use these ideas to create a complementary product and other uses which directly lead to the discovery of the same ideas and principles, but then indirectly show the way to the creation of a piece of software which is competing with the decompiled one? If we do not decompose the analysis in an access phase and a re-implementation phase, in the field of APIs the transformative v. substitutive uses distinction becomes a

---

[214] As reported by NAKANO & HIRAKAWA, *Computer Interfaces in Japan*, p. 50.

[215] Id., p. 54.

[216] For instance, Nakano and Hirakawa notice that "the definition [of programming language contained in the JCL] does not appear to suggest that the letters or symbols embody the program, only that they be used as a method to express the program." Starting from this observation, they argue that macros and other "application specific languages" are copyright protected, since they "embody" program functions, instead of just being a tool of expression. Id., 54. In my opinion, there is no basis for this statement. In fact, in any (high level) programming language, several single words actually "embody" a lot of code. That is true because each word in a programming language may be used to recall libraries of functions, which have been written by the developers of the programming language (*rectius*; which have been written by the developers of a given implementation of the given compiler used to transform in object code the program at hand). Hence, I argue that also "application specific languages" are not protected by copyright in Japan. However, it is clear that almost each "word" in these "languages" is actually a shortcut to call some code implementing functions at a lower level; and this code, as long as it is not technically determined, constitutes protected expression.

[217] The strict analogy between programming languages and APIs has been observed also by DAVID S. EVANS, et al., *Invisible Engines -- How Software Platforms Drive Innovation and Transform Industries*, (David S. Evans ed., MIT Press First paperback ed. 2008).

[218] Definition of Programming Language as found in the Wikipedia, http://en.wikipedia.org/wiki/Programming_language.

[219] The analogy between APIs and programming languages is especially clear in the case of object-oriented programming (see http://en.wikipedia.org/wiki/Object_oriented_programming). In this context, for instance, one may create objects that are functionally fungible (in terms of their interaction/interoperation with the rest of a program), even though they are created with completely different code.

distinction between vertical and horizontal interoperability. The likely effect – again, if we do not decompose the access/re-implementation phases[220] – is that we may end up, as some authors,[221] recommending to treat in different ways the same kind of reimplementation, depending on the competitive relationship between the original developer and the decompiler/re-implementer.

According to Weiser, in particular, there is a "critical distinction" between "horizontal and vertical access".[222] To be sure, I will not try to deny that it is generally true in economic terms, since horizontal access is equivalent to direct competition between two agents, while vertical access is equivalent to complementarity and no (or – at most – merely potential) competition. What I will try to show is that this difference is not so directly linked with what Weiser recalls next, that is the Supreme Court's decision in Campbell v. Acuff-Rose. In that ruling, the Court "suggested that the fair use analysis should take a [...] nuanced analysis by evaluating whether the use of copyrighted material serves a 'transformative' (and therefore permissible) or a 'substitutive' (and therefore impermissible) purpose".[223] Weiser seems to argue that only vertical interoperability may constitute a transformative use. In fact, I argue that the distinction highlighted by the Supreme Court concerns the direct purpose of uses to be evaluated as fair or not, while Weiser proposes to analyse the competitive relationship existing between the original decompiled software and any software, the creation of which may be helped by any use of the original software. In other words, if there were no means to distinguish between access to an API specification and creation of an API re-implementation, then Weiser would be completely correct and his quotation of Acuff-Rose perfectly appropriate. In that case, using a copyrighted work to attain vertical interoperability would fail a fair use test, while a use that "competes" with the original product would be generally less likely to be "fair". Alternatively – if we do not want to decompose access and re-implementation, but we still do not want to deny horizontal interoperability as well – we may try to be very flexible (should I say sloppy?) in defining transformative uses, as happened in Sony v. Connectix,[224] which is a case severely criticised by Weiser and – in my opinion – for sound reasons, since the Court is reaching an appropriate result (or, at least, the one I suggest), but in an improper way. In fact, in Sony v. Connectix the defendant re-implemented Sony's APIs in a PC software, so that (original or pirated) videogames realized for Sony's (PlayStation) console could run also on personal computer (gaining horizontal interoperability, although not perfect and with lower performances). According to Weiser,

> "[t]he Connectix court, in an attempt to harmonize Sega[225] and Campbell, concluded that the Virtual Game Station was 'modestly transformative' because it constituted a 'wholly new product, notwithstanding the similarity of uses and functions between the Sony PlayStation and the Virtual Game Station.' In so doing, it failed to distinguish between cases like Sega, where the provider of the application designed a complementary product, and those like Connectix, where the purpose of interoperability was to compete with the underlying information platform."[226]

Hence, as I already admitted, if we could not distinguish between access to and use of interoperability information, I would agree with Weiser. Horizontal access (to produce a substitute of Sony's console) could hardly constitute a fair use, and it would be so mainly because of the last step of the fair use analysis, that is because of its (potentially significant, even if practically negligible, with some insights) negative impact on Sony's sales.[227] To decide otherwise would need peculiar arguments, like the one of the Ninth Circuit, which concluded that Connectix's product was "modestly transformative", which is a far from clear concept.

---

[220] To be sure, I will suggest the application of a fair use analysis to the access phase (involving decompilation), but not to resort to fair use (or Civil Law exceptions and limitations, like Art. 6 of the *Software Directive*) in the re-implementation phase.

[221] See, in particular, WEISER, *The Internet, Innovation, and IP Policy*.

[222] The same point had been stressed by K. W. DAM, *Some Economic Considerations in the Intellectual Property Protection of Software*, 24 The Journal of Legal Studies, 321--377 (1995), using the wording "attachment" (substantially coincident with vertical access) and "replacement" (horizontal access).

[223] WEISER, *The Internet, Innovation, and IP Policy*.

[224] Sony v. Connectix, 203 F.3d 596.

[225] The reference is to Sega 977 F.2d 1510 (9th Cir., 1992), a vertical interoperability case.

[226] WEISER, *The Internet, Innovation, and IP Policy*.

[227] I am aware that this is just one (the last one) of the steps of a fair use test (and that no single step in this test should be automatically considered to prevail over the others), but its role in this case is very relevant and the other steps are not likely to counterbalance it. Weiser proposes to proceed in this way: "Under the competitive platforms model for regulating access to information platforms, the Ninth Circuit should have accepted Sony's claim of infringement. In particular, the court should have looked more closely at whether the markets for video games and computing were in fact converging, such that VGS's product would compete with Sony's. Because there are important signs that this was the case (as recognized in the district court's opinion, for example), the next question would be whether Sony enjoyed sufficient market dominance to justify access to its platform to create a rival one along the lines of the VGS. Given Microsoft's recent entry into this market, and the continued strength of

However, if we decompose the access and the reimplementation phases, we discover that the access phase starts from a protected API implementation and creates a new work, that is an API specification, and this is always a highly transformative use. To obtain access to the ideas and principles needed to write a meaningful API specification document, unfortunately, formal copyright infringements are performed and unauthorized derivative works are created. Hence, a fair use exception is indeed needed, because infringing works are created and used as "intermediate goods" (i.e. as versions of the original work which are easier to study and understand) during the access process. That said, the "final good" of the access phase (i.e. a specification manual or simply some notes allowing a better understanding of the working of the original software) are surely new, original and highly transformative goods. Moreover, when we speak about the reimplementation phase, the wording "transformative uses" may be even misleading: non-infringing re-implementations of APIs are not only transformative uses; they are completely new immaterial goods. They are like the same biography told with other words, in a context in which the story is not fictional, but historically determined. In other words, the access to interoperability information and the creation of a new interoperable product are not a "unique use". Actually, only to understand interoperability information one needs to access and "make use" of the original product: during this phase, it is granted that – without a finding of fair use – several illicit copies of the original software must be made. That having been said, if we accept to decompose the analysis, one may reach the same conclusion as the Ninth Circuit, without recurring to misleading (muddy?) concepts like the one of "modestly transformative use". In fact, Connectix could have produced only a manual called "Conditions for a software platform to run games produced for Sony's Playstation" (reconstruction of an API specification) and a third party could have implemented these conditions (independent implementation of the reconstructed specification). In this case, the fair use analysis should have concentrated on Connectix (virtual) manual, which would have been clearly transformative (and obviously non competing with Sega's product) and could have beneficiated of a fair use exception.

Differently from what is suggested in this paper, Weiser urges to examine copyright cases involving reverse engineering and interoperability issues performing a "flexible inquiry" and adopting "a competition policy focus". For instance, he suggests that

> "where a first mover [in the spreadsheets market] like Lotus already had received a 'substantial reward for being first', which appears to have been the case in Lotus, intellectual property protection should recede and allow others to appropriate some of the value of the industry standard so as to allow for competition".[228] [More generally, in the approach proposed by Weiser to deal with interoperability issues in the real world,] "the legality of reverse engineering or copying a user interface should follow a three-part inquiry. First, it should consider whether the inventor, through first mover advantages and the like, has reaped a sufficient reward […][229]. Second, it should evaluate whether competitors could challenge the proprietary standard's position in the market without such a sharing of the user interface. Finally, it should determine whether the company seeking to take advantage of the sharing requirement used it to facilitate the introduction of a differentiated rival product rather than merely imitate the initial invention."[230]

If feasible, Weiser's approach would likely be optimal, as any elaborate "rule of reason" based on a rich economic analysis.[231] The problem with it is that judges should be able to do the following: understand all the

---

Nintendo, it seems that no such access is necessary, and thus Connectix's copying should not have been judged a fair use. Finally, if Connectix had engaged in the reverse engineering of a dominant standard, it should also have been required to show that it did not merely clone Sony's product, but added some value to it."

[228] WEISER, *The Internet, Innovation, and IP Policy*.

[229] "In nearly all cases where an information platform captures a dominant share of a market, this consideration will weigh in favor of allowing open access"… and – one may ask – why do we not use antitrust if this approach is particularly appropriate in the case of dominance or quasi-monopoly? In fact, antitrust has the problem of being an ex post analysis, but also such a flexible application of IP risks to be ex post: in both cases, if the criteria are clear, firms should be able to "solve the game by backward induction" and avoid violations of the law (unless it is economically convenient to violate it, as it is frequently the case, as Microsoft demonstrated – not only with repeated antitrust violations but also paying huge fines to the EU just to delay the implementation of antitrust decisions).

[230] A similar approach is suggested also by OTTOLIA & WIELSCH, *Legal Aspects of Modularization and Digitalization*: "[T]he owner of a dominant standard in a platform, like Microsoft for the operating system, cannot prevent others from accessing its interfaces when they engaged in building compatible programs, but […] he is protected when somebody uses the access to interfaces just to build an imitating product without any functional surplus."

[231] For instance, Weiser also suggests to apply a similar rule of reason approach in the field of user interfaces: "[t]he flexible approach animated by the competitive platforms model recognizes that the ideal form of competition would come if other providers could successfully offer an alternate—and superior—form of a user interface. By contrast, if there was only one

possible competitive strategies and technical possibilities available to competitors; decide if a given strategy is "worth" a limitation to IPR or not; and finally de facto tailor the duration of IPR case by case. It is clear that – if we assume that judges are able to do this at reasonable cost – they should be encouraged to do so. But one may doubt that judges (or other human beings) could actually be able to perform this kind of analysis. Moreover, in thinking about the practical feasibility of Weiser's approach, one should also consider that, unfortunately, it is dramatically easier to see the "right decision" ex post, than to decide when it is time to really write a ruling. Authors trying to support this kind of approach using case studies have a quite easy time, in comparison with courts actually facing the specific and actual case. In fact, if you look at cases a few years after the decision (or even after a few years from the beginning of the case), the "right" decision to influence, in the best way, the development of technology looks frequently obvious, but it usually was not so evident in the first place. In addition, since the author seems to require (at least potential) dominance in order to allow for a "free riding on interfaces", why cannot we directly use competition policy, without having a blurred notion of intellectual property rights on interfaces, determined case by case?

In a few words, Weiser's approach is victim of the three criticisms that the author already knew of (but to which he did not reply in a completely satisfactory way): difficulties of distinguishing between vertical and horizontal access; administrative costs; and – what is more important – difficulties in determining "at what point in a standard's development a rival firm should be permitted access." I already hinted at administrative difficulties and I will mention the theoretical difficulty of distinguishing between vertical and horizontal access in a few lines, but now I want to highlight that – even if Weiser's article seemed to take into account the fact that decompilation is not a perfect tool to get access to interfaces – the conclusions of the article did not seem to take this fact into account in the appropriate way. As suggested by Samuelson, Scotchmer, Johnsons-Laird and many others,[232] I think that the imperfection of decompilation (reverse engineering) is so strong that it should always be allowed because there are no known cases in software industries where decompilation actually allowed for enough free riding to undermine the leadership of a dominant firm (if not after a lot of time and coupling decompilation with a lot of other investments generating a new, better product). My suspect, *en passant*, is that the cases in which Weiser's article suggest to *allow decompilation* are actually the ones in which (taking the suggested precaution concerning incentives to innovate) it could be sensible – for competition policy authorities – to *mandate disclosure*. Indeed, when we deal with vertical access and one player if preventing another from achieving it, the approach proposed by Weiser could be used in order to decide whether we should "mandate disclosure" of the interoperability information. And this is a quite different problem with respect to crafting intellectual property in a way allowing self-help of late comers through reverse engineering. (The third paper of this dissertation project will touch upon this different problem, in § 4.6 and 4.6.4 in particular.)

Finally, if I argue against Weiser's suggestion that vertical and horizontal interoperability should be treated in different ways it is also because (1) it is not easy to clearly distinguish between these two categories and (2) because there are various degree of horizontal interoperability and the one achieved through reverse engineering is not likely to allow the creation of perfect substitutes (and – for this reason – it is not likely to trigger disruptive competition and market failures). About various types and degrees of interoperability – when a leader commercial software house tends not to favour (or actively discourage) interoperability – it is commonplace that software programs are not able to perfectly interoperate with other software of the same generation: "For instance, the Samba server software producer is more than one generation late in the reverse engineering of Windows interfaces. Furthermore, success depends on the platform operator not changing the code; this could easily be done, even through legitimate upgrading of the code".[233] At the same time, major changes in the code of the platform may impose significant costs on users, so that this strategy cannot be used in an excessively free way and backward compatibility is usually guaranteed (at least for basic functions).[234] Systems which are complex enough and which evolve at a sufficient speed, providing a constant stream of new APIs, are quite safe against reverse engineering, while soon or later, if a dominant system is

---

appropriate form of a user interface (or it appeared that one would ultimately emerge as dominant), that single usable interface should not be protected."

[232] SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*; JOHNSON-LAIRD, *Software Reverse Engineering*.

[233] ROTENBERG, *Regulation of Software Interoperability*.

[234] Moreover, arbitrary changes in the interfaces of a dominant platform would likely be subject to antitrust scrutiny, especially if they impose costs on users without bringing substantial advantages.

static, it will be decompiled successfully and functionally cloned.[235] This scenario should not be worrying for software developers, even if it is obvious that dominant actors could prefer an almost complete ban on decompilation. On the one hand, the possibility of reverse engineering puts additional pressure on leaders to innovate (in some cases maybe with some distortions against backward compatibility, which could represent one of the main cost of relatively easy reverse engineering). On the other hand, if the law increased too much the cost of reverse engineering, it could eliminate the possibility of using it to win against an incumbent innovating at a very slow pace (and living only on the rent generated by the combination of various types of network and learning effects).[236]

8. Elimination of free riding vs. the creation of economic monopoly

It is now time to compare the legal and technical findings of the paper with a sketched economic model of the legal protection of software works, also taking into account the other economic insights I outlined in the *General Introduction* to the dissertation. I can anticipate that – if applied as I suggested – the so-called "technology copyright"[237] model not only works in a reasonably sensible way, but it is much more consistent with economic insights than alternative, more protectionist, models. In fact, a higher level of protection potentially offered by an alternative patent-like model would likely raise several problems, without solving major incentive problems. Indeed – as I will try to show – major market failures due to the lack of incentives do not exist even in a more flexible copyright-based model. As I discussed in the *General Introduction*, I will start from the idea that the typical condition for an efficient market to work is not that producers are able to appropriate the entire social benefit generated by their activity. Such a condition has an interesting property: it ensures that – in a static setting – each and every socially profitable investment is performed. However, it does not consider the huge transaction costs generated by a capillary system of exclusive powers, able to extract the entire surplus generated by a given innovation. What is sufficient to have an efficient market is that investors are able to recoup their costs (taking into account risk), as long as consumers appreciate their products. Obviously, in a market in which there are investments and fixed costs, this condition cannot be respected by the equality among price and marginal costs. Instead, it is verified when "gross profits" are equal to sunk costs.

As long as reverse engineering is not prohibitively difficult and/or hindered by the law (a problem that I will address in the second paper of this dissertation), coupling copyright (preventing literal copying and other form of parasitism) and trade secret (giving limited but significant protection to some categories of innovations) could create a structure of incentives working as a quasi-liability system in the field of software interfaces.[238] This system avoids two polar and opposite risks of the patent systems. On the one hand, rightholders cannot block subsequent innovation for strategic reasons and self-help remains an available solution also in cases in which transaction costs (frequently increased by strategic behaviour and/or asymmetries of information) would have prevented the working of a market. On the other hand, significant incremental innovation, which would not qualify for patent protection, is still protected against easy and cheap appropriation from third parties. A quotation from Prof. Reichman's seminal 1994 article may be useful in clarifying the role of trade secret as a stimulus for innovation:

> "Legal theorists have particularly underestimated the important role of trade secret laws (or equivalent laws of confidentiality) in mediating between formal intellectual property regimes and free competition. These laws do not confer exclusive property rights in the manner of patent and copyright laws, but they do require would-be competitors to extract an innovator's undisclosed know-how by proper methods of reverse engineering.[…]
> Second comers who cannot extract valuable undisclosed information by proper means or independently reach similar solutions must acquire the unpatented know-how through licensing agreements with

[235] For instance, I am not aware of specific data concerning the proportion of application designed for Windows 95, 98, XP or Vista that may run under Linux or Mac using Wine, but this could be an interesting illustrative example and my guess and personal experience is that it is fairly easy to run the oldest one without a copy of Windows 95 – but, obviously, this does not pose a significant competitive threat to Microsoft, since these old application can be considered quite obsolete.

[236] See the second paper of this dissertation project for a fuller discussion of these issues.

[237] I will put a higher emphasis on copyright (instead of patents) because this is the main tool of actual protection of software related innovation in all legal systems I am aware of. However, I will briefly touch problems related to patent protection of software in *§ 9.2. May patent law (as currently applied to software) limit interoperability?*.

[238] *See* J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Columbia Law Review, 2432 (1994) and several quotation in the General Introduction and below.

> innovators. Either way, these legal requirements normally provide those who develop unpatented, noncopyrightable innovation with a period of natural lead time in which to recover their investments while establishing their reputations as producers of quality goods."[239]

In fact, I have to stress that Prof. Reichman does not believe (or, at least, did not in 1994) that trade secret is likely to work as described above in the field of software. Indeed, in this and in similar fields, Prof. Reichman thought that innovation is too "near to the surface" of products and risks being easily appropriated, so that the protection offered by secret would be insufficient. Hence, he proposed that the law should reinforce the natural lead-time of innovators, exposed to an excessively cheap reverse engineering:

> "In dynamic economies driven by constant technological innovation, competition with respect to the products and processes of routine innovation thus presupposes a degree of natural lead time, which classical trade secret law is presumed to supply. Absent natural lead time, however, investment in innovative but unpatentable applications of science to industry tends to dry up, and competition may languish in the face of a progressive market failure. As will be seen, the likelihood that this type of market failure will occur has greatly increased during the second half of the twentieth century."[240]

Despite the fact that this influential article was very likely correct in its theoretical analysis, I suggest that it was slightly less well-grounded in terms of empirical and technical evidence. And what happens in the "real world" in crucial in drawing the consequences of Prof. Reichman's theoretical work, in fact:

> "The term "natural lead time," though analogous to the economists' "head start," […] is chosen to emphasize the extent to which implementation of the governing legal regimes -- trade secret law and the law of confidential information -- depends upon real world events and not legally determined outcomes, such as a fixed term of duration. Natural lead time depends on the individual business decisions and strategies of both innovators and borrowers, not on the dictates of legal rules."[241] [And, I would add, it crucially depends on the state of technology.]

In fact, I do not think that reverse engineering in the field of software is so cheap that it is likely to create market failures, and I will try to show this in the second paper of this dissertation. Moreover, in that paper I will also show that the law already reinforced significantly trade secret in the software field, hindering reverse engineering. Actually, I will even argue that this reinforcement is excessive and that it could create some market failures similar to the one of a strong patent system; however, should I be wrong, one could see this reinforcement of trade secret in the field of software as an application of Prof. Reichman's analysis.[242] Overall, having different assumptions concerning the cost of software reverse engineering, I share the theoretical analysis of Reichman, but the conclusion I reach about software is almost diametrically opposed to his own. In fact, this difference very probably derives from the fact that Prof. Reichman was analysing innovation in the field of new technologies in general, while I am focusing on a specific kind of technical innovation (interface information), which is much easier to "keep secret" than the average software innovation. In other words (and using extreme examples), if one invents a "word processor" it is evident that this is innovation so "near to the surface" of the product that it can be easily appropriated by other players. The same may be true for innovations concerning user interfaces in general (because these elements are inherently "near the surface" of the software product). However, if one creates an innovative set of APIs the fact of just distributing the humanly not understandable object code, coupled with some appropriate non-disclosure agreements with producers of complementary products, may be very effective in giving years of lead-time to the first comer (easily 5 or 10 years, as some examples will show in the second paper). Thus, at

---

[239] Id., pp. 2438—2439. See also pp. 2440—2441: "On the margins of the pure market economy envisioned by nineteenth-century liberal economic thought, trade secret laws (and related laws protecting confidential information) thus provide a loosely constructed set of liability rules that reinforce the competitive ethos in subtle and indirect ways. These modified or quasi-liability rules mediate between the potential for overprotection inherent in the statutory grants of exclusive property rights and the potential for underprotection inherent in the competitor's unfettered ability to appropriate the fruits of investment in unpatented incremental innovation."

[240] Id., p. 2442.

[241] Id., f.n. 25.

[242] Actually, the proposal of Prof. Reichman would be much more sensible, from an economic point of view, of the generalised reinforcement of trade secret in the field of software that one may observe in modern legal systems. In more practical terms, the quasi-liability system proposed could "provide a maximum blocking period of [two, three or four] years, during which time competitors could not enter the same market with substantially the same unpatented product absent an agreement with innovators" (Id., f.n. 522 and accompanying text.). In the field of APIs this could mean that horizontal interoperability could not be achieved by reverse engineering for a certain period of time (during which – however – study and experimentation would be free).

least in the field of software interfaces, it remains possible to use property rights (and copyright in particular) – with their innate pro-market characteristics – instead of creating special liability rules.

Incidentally – and even if I did not (and will not) discuss the possibility of *sui generis* protection for software[243] – the first part of the present work made clear how precious the general principles and doctrine underlying copyright law are in order to tackle several complex and cutting edge issues arising in the field of the legal protection of computer programs. An *ad hoc* model of protection would risk lagging constantly behind technological innovation and would likely provide minor advantages, with respect to copyright, as long as copyright is not excessively expanded (in the direction of a quasi-patent right) and it is interpreted considering the technological reality, as I suggested in the paper at hand.[244] For this reason, I share the conclusion of several authors[245] arguing that the desirability of this kind of *ad hoc* protection (even though theoretically sustainable as a first best solution, tailored for software) raise several serious questions, mainly concentrated on the lack of experimented and shared legal principles.

In summary, I clearly agree with Prof. Reichman when he argues that "the nineteenth-century vision that subdivided world intellectual property law into discrete and mutually exclusive compartments for industrial and artistic property has irretrievably broken down."[246] However, the breakdown of this divide does not mean that we should resort to a high number of *sui generis* paradigms. The patent and copyright paradigms remain perfectly valid and both useful depending on the kind of innovation that they have to protect (at least as long as they are interpreted as I suggested in the *General Introduction*) and even if copyright is no longer used just as a tool to protect artistic works. In fact, Reichman argues that software and other subject matter protected by "hybrid legal institutions", such as plant varieties, "violate the negative economic premise that limits copyright protection to cultural goods".[247] As I tried to show in this paper (and as I will discuss further in the second one), I argue that there is not such an "economic premise", at most there is an "historic premise" saying that copyright protects just aesthetic creation. Economically, the fact of protecting just expressive form and not the underlying ideas is not necessarily an aesthetic category: in the field of software, this means that the general structure, purposes and methods of a program are not protected, while the actual code is. And writing this code is a very significant line in the budget of a software house (that I would label "development cost", as opposed – but complementary – to pure "research costs"). To be sure, the "historic premise" according to which copyright concerns artistic creations has some consequences (as the already mentioned and unreasonably long duration of copyright protection), but – in general – copyright remains a good tool to protect all those kind of innovative sectors in which Development costs are significantly higher than Research costs (in the sense I already explained), so that a partial free riding on ideas and principles is not capable of creating significant market failures.[248]

Finally, I acknowledge that adopting the interpretation of APIs' legal protection proposed in this paper may seem to largely disregard incentives to realize interfaces. Yet, I already clarified – and the second paper will make even clearer – that, because of the cost of reverse engineering software, there is actually a quite broad scope for the licensing of API specification documents (also under non-disclosure agreements). In any case, I concede that the kind of incentive to create that comes from the possibility of licensing specification documents is not very similar to true property rule. Instead, it is based on something similar to a quasi-liability rule. In other words, licensing API specifications in a world where reverse engineering is allowed and copyright does not protect specifications is equivalent to "selling" (substitutes of) the reverse engineering

---

[243] About *sui generis* protection of software, see PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, see id., 2308--2431 and the debate generated by the *Manifesto* (see, in general, the Columbia Law Review issues of 1994: JANE C. GINSBURG, *Four Reasons and a Paradox: The Manifest Superiority of Copyright over Sui Generis Protection of Computer Software*, 94 Columbia Law Review, 2559--2572 (1994); PAUL GOLDSTEIN, *Comments on a Manifesto Concerning the Legal Protection of Computer Programs*, 94 Columbia Law Review, 2573 (1994); PETER S. MENELL, *The Challenges of Reforming Intellectual Property Protection for Computer Software*, 94 Columbia Law Review, 2644 (1994); ZENTARO KITAGAWA, *Comments on 'A Manifesto concerning the Legal Protection of Computer Programs'*, 94 Columbia Law Review, 2610--2620 (1994)).

[244] See BRETT A. CARLSON, *On the Wrong Track: A Response to the Manifesto and a Critique of Sui Generis Software Protection*, 37 Jurimetrics J., 187 (1997).

[245] See, in particular, GINSBURG, *Four Reasons and a Paradox*. Several of the authors mentioned in footnote 243 and criticizing the *Manifesto* share similar conclusions.

[246] J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, see id., 2432, 2500.

[247] Id., 2502.

[248] And – as I already explained – this is even truer in the field of software, given the advantages of first comers (direct and indirect network effects, learning costs and other switching costs, etc.) and the cost (and imperfections) of software reverse engineering.

activity needed to discover interfaces, not to license the interface specifications themselves.[249] Moreover, even assuming that reverse engineering would be so simple as to destroy direct incentives to innovate (which is an absurd assumption at present), alternative sources of incentives to the realization of good interfaces abound. Actually, the majority of advanced two-sided models concerning software platforms (i.e., the kind of programs that typically expose more interfaces) suggest that controllers try to subsidize developers of complementary software;[250] this implies that there is actually a strong incentive, in order to maximize the value of a platform, to develop and give away for free high quality interfaces. Obviously, problems remain about competing software (i.e. horizontally interoperable software), but here an empirical example may be useful. After all, several open source software are perceived as being of very high quality – in some cases even better than their commercial counterparts – but this does not allow commercial firms to develop their one competing software at no cost (unless they distribute themselves as open source under the GPL or similar licenses). Why so, if knowing the source code of a competitor is enough to free ride on it? Probably simply because that is not enough at all, and well-written software does make the difference, even when the ideas behind it are free to be taken.

8.1. The limits of technology copyright and its natural antibodies

Of course, basing the creation of incentives to write new and good software interfaces on a *de facto* quasi-liability rule, deriving from the combination of copyright and trade secret (what I call "technology copyright"), is not a panacea and it entails several costs. In particular, Prof. Reichman observed that

> On the negative side, this substratum of modified liability rules entails appreciable social costs of its own. It is also singularly prone to yield arbitrary and irrational results whenever the task of reverse engineering unpatented, non-copyrightable innovation proves either too difficult or too easy.[251]

Differently from what Reichman expected (in 1994), this task normally proves too difficult – and not too easy – in the field of software (and I will discuss more about that in the second paper of this dissertation). Moreover, the actual working of the industry, coupled with copyright, prevents "incremental innovators" from taking someone else's software as a basis for writing new software. In this way, new bugs are introduced every time a piece of software is developed and it is quite frequent for developers to "reinvent the wheel". In other words, "technology copyright" is costly, since it slows down the spreading of innovation (and indirectly generates waste through reverse engineering).[252] At the same time, even copyright alone would probably not be a perfect tool to protect software. In fact, it allows developers to learn from others (as long as they can access source code), but then forces them to start from scratch every time they write an application (frequently they do not even use their own past pieces of code, because they are owned by past employees). A specific analysis of these problems has been proposed by Lemley and O'Brien in an article appropriately titled "Encouraging Software Reuse".[253] Also Determann[254] highlighted this problem, stressing the faults of copyright in determining this outcome:

> "Since courts first decided to afford copyright protection to computer programs, commercial software development companies have had a strong incentive to avoid reusing existing code owned by others. Independent creation is a defense to copyright infringement, and so software development companies often opt for creating programs from scratch, ideally in a "clean room" environment, so they can prove that their products are not copies of existing programs with similar functionality. Thus, the decision in favor of software copyrightability had a rather dramatic impact on the professional lives and day-to-day activities of programmers: instead of being asked to further develop and improve the "state of the art" and to focus on cutting-edge problems, programmers were asked to spend most of their time reinventing

[249] But one has to be aware that it is possible to directly license interface *implementations*.

[250] *See* J. C. ROCHET & J. TIROLE, *Two-Sided Markets: A Progress Report*, 37 RAND Journal of Economics, 645--667 (2006) (or J. C. ROCHET & J. TIROLE, *Two-Sided Markets: An Overview*, IDEI Toulouse working paper (March, 2004)).

[251] REICHMAN, *Legal Hybrids*, p. 2441. Cf. also LOTHAR DETERMANN, *Dangerous Liaisons -- Software Combinations As Derivative Works? Distribution, Installation, And Execution Of Linked Programs Under Copyright Law, Commercial Licenses, And The Gpl*, 21 Berkeley Technology Law Journal, 1421 (2006), p. 1437: "copyright law strikes a delicate balance between access and exclusion rights", so that "[b]oth under- and over-protection can harm the public interest in creative works."

[252] *See* RICHARD R. NELSON, *Intellectual Property Protection for Cumulative Systems Technology*, 94 Columbia Law Review, 2674 (1994), p. 2676.

[253] MARK A. LEMLEY & DAVID W. O'BRIEN, *Encouraging Software Reuse*, 49 Stanford Law Review, 255--304 (1997).

[254] DETERMANN, *Dangerous Liaisons*.

> the wheel. Why? Because lawyers did not have the energy or wit to come up with a more fitting intellectual property law regime tailored to software."[255]

That is surely true; however, any tool generating incentives is also likely to generate social costs. (Alternative, but still significant, kinds of costs would be generated by patents, in the form of costs of inventing around and/or transaction costs and strategic risks to deter follow-on innovation.)

Overall, the problem is mainly to choose what kind of costs one wants to pay in order to spur investments. To be sure, a system pushing to "reinvent the wheel" is a costly one. Nevertheless, it is precisely in responding to these kinds of difficulties that copyright proved to be an appropriate tool to protect software because of its flexibility. In fact, there is another element of copyright law applied to software that Reichman could hardly have fully understood in 1994 (but which is relevant and should be taken into account, with insight): a nice feature of the copyright paradigm is that it allows developers to recognize cases in which the afforded level of protection is excessive for the industry as a whole and allows them to easily (and cheaply) contract around this potential market failure.

In particular, the Open Source movement – as Determann[256] already recognized – can be seen (also) as a response to the costs generated by copyright.[257] (To be sure, the Free Software movement, from which the broader open source approach originated,[258] has not been born to solve any market failures. Instead, it is based on philosophical and moral considerations, not on economic efficiency.[259] However, I doubt that open source could have been successful also as a business model, if it had not been also able to solve some inefficiencies of the traditional proprietary model of software development.) Under the open source paradigm, software programmers can improve and correct what exists, instead of starting from scratch every time and this gives them significant advantages over traditional developers. However, this model too comes with a cost, that is represented from the need for creating new business models to (indirectly) generate incentives, since direct monetary compensation for writing code that anybody could appropriate is unlikely (and in any case insufficient). (In other words, economic incentives to innovate, in this model, come from "second order" phenomena, like revenues associated to any good or service complementary to the produced software products. Again, to be sure, incentives also come from the simple pleasure of creating and sharing, as from the intellectual enjoyment of solving bugs and improving a common resource: I hope that open source supporters will pardon me, if I just focused on more prosaic aspects of this approach to software development.) Discussing the open source model in general and its incentive structures would stretch too much away from the main topic of this paper.[260] In any case, this approach to software development is by now known to the public, and it is easy to understand why I argue that the existence of copyleft (along with its success) is one of the best demonstration of the value of copyright to protect software. In fact, copyright is cheap and flexible enough to be turned into copyleft. Applying (directly, for once) Coase's theorem, one may say that copyleft, so to speak, "contracts around" copyright fallacies.[261] To allow this system to work, however, certain rules of reciprocity need to the enforced. Some of these norms are socially created, at the level of a community of developers and users, accepting some common principles of sharing. Other rules are legally enforced – especially against free riders outside the community – precisely through copyright, thanks to the viral clauses of several copyleft licenses.[262]

---

[255] Id., 1480.

[256] Id..

[257] *See* also STEPHEN M. MAURER & SUZANNE SCOTCHMER, *Open Source Software: The New Intellectual Property Paradigm*, NBER Working Paper No. 12148 (March, 2006), p. 4: "The open source movement emerged to support an industrial product (software) for which disclosure of code is particularly useful, but not required by intellectual property law."

[258] The term "Free Software" relates more to the freedoms this approach gives to users and developers; the wording "Open Source" explicitly focuses on the technical aspects related to the availability of the source code and on the related strengths of this software development model. About the differences between Free Software and Open Source, see also the Wikipedia page http://en.wikipedia.org/wiki/Free_and_open_source_software.

[259] See, in particular, the works of Richard Stallman, who may be considered as one of founders and as the spiritual father of the Free Software movement. A short biography and several references may be found starting from the Wikipedia page devoted to Stallman (http://en.wikipedia.org/wiki/Richard_Stallman).

[260] However, I share several of the points of the following paper, to which I remand: JOSH LERNER & JEAN TIROLE, *Some Simple Economics of Open Source*, 50 The Journal of Industrial Economics, 197--234 (2002).

[261] The so-called Coase Theorem suggest that – in a setting with well defined property rights and low transaction costs – private parties would allocate entitlements to use or not to use resources in an optimal way, and so did – according to my interpretation – the open source developers.

[262] In particular of the most common of these licenses, the GPL one (see the GNU project website, in particular http://www.gnu.org/copyleft/).

Notice that the existence of a phenomenon like the open source movement is an additional reason for which I strongly prefer copyright to patents as a tool to stimulate software innovation. People accustomed to the working of the patent systems immediately see that creating a "patentleft system" would have been very difficult, if not impossible, in particular because of the significant transaction costs related to patents, starting from the filing cost itself. Compared to patents, a copyright-based solution has the advantage of establishing property rights at a very low cost (there are no formalities and hence no costs in claiming copyright) and of allowing very easy transactions concerning the conferred rights, through the very well established tool of copyright licensees. Notice also that the open source approach makes more relevant some features of copyright law, which are normally neglected in the field of software, like the fact that derivative works are themselves copyright protected creations. Once one has the right to borrow from others (at certain conditions) he/she may also create new, protected, works. Hence, as observed by several authors,[263] "[i]nnovators may, indeed, protect selections and arrangements of information as copyrightable compilations to the extent they contain original and creative expression." Finally, even though the open source model is not necessarily the best suited for every kind of project and situation (or – at least – it is far from proven that it is), a great advantage of this approach is that it can perfectly coexist (at least in principle) with the commercial model of software development. Copyleft is the opposite of copyright from the philosophical, moral, economic point of view; however, both models of software development are legally based on copyright (used in very different ways).[264]

9. Further dimensions

In this section, I touch on several points, which are relevant to the topic of legal protection of APIs, but are not strictly focused on copyright protection. Considering them is useful, in order to provide more robustness to my tentative conclusion that copyright – coupled with trade secret and in an environment where reverse engineering is legitimate – may provide an optimal tool of protection for software interfaces. (This optimality, of course, is not absolute, but relative to the comparison with alternative tools of legal protection or with alternative interpretations of copyright law.) In section 9.1 and its subsections, I analyze the possibility that contractual arrangement could empty of any meaning the careful balancing exercise performed in interpreting copyright law in a sensible way for the software market. As an antidote to this risk, I argue that courts should embrace more explicitly the doctrine of copyright misuse and/or that some clauses of shrink-wrap or click-wrap licences (i.e. of essentially "take-it or leave" offers) should be considered preempted by copyright law. Section 9.2 will discuss another important source of risk menacing to disrupt the aforementioned balance of interest: the possibility that patent law, as currently applied to software, may significantly limit interoperability. Finally, in section 9.3, I will discuss (and try to confute) a class of potential economic critiques to my approach, based on the fact that there are good reasons for which a platform producer may want to manage interoperability (which entails some control over specifications, not only implementations).

9.1. Contractual arrangements

In section 8.1, I showed that copyright licenses and other contractual arrangements are used to avoid some of the market failures that can be generated by the combination of copyright and trade secret. However, copyright licenses could also be used to create new market failures. Indeed, the general conclusions of this paper about the appropriateness of protecting software interfaces through copyright and trade secret would have little or no empirical relevance if it were possible for copyright holders to use standard software licenses "contracting out" the possibility of accessing or using interoperability information. That is the case, because – in order to obtain access to interoperability specifications – one should usually become a licensee of the original software (at least as a final user) and accept the associated licenses. Hence, the licensor may impose

[263] See, in particular, REICHMAN, *Legal Hybrids*, pp. 2545—2546 and also f.n. 22 for additional references.

[264] Eben Moglen (law professor at the Columbia University, and one of the main authors of the GNU GPL license, which is the most common open source copyleft license) frequently recalls in his speeches that the more copyright is reinforced, the stronger copyleft becomes. This is true both from a dialectic point of view (in the sense that an "oppressive copyright" may reinforce the support for copyleft and the perception that an alternative is needed) and from a legal point of view, in the sense that licenses such as the GPL become easier to enforce in a legal system granting broader powers to rightholders.

on the licensee standard licenses (in particular "shrink wrap" and "click wrap" licenses),[265] which could be used to contractually limit the freedom to reverse engineer, create competing or complementary products, replicate interface specifications and so on.

9.1.1. Licenses and copyright misuse

Standard license may disrupt the careful balance of interest devised by what I labelled "technology copyright". Opportunely, there are several ways to prevent these limitations from being legally bounding and enforceable: I will not discuss those based on general contract law[266] or antitrust law[267] (or consumer law, if applicable).[268] Instead, I will concentrate on an "interne antibody" of intellectual property law: copyright misuse[269] (or equivalent civil law approaches, if existent at all).[270]

In the Atari case[271] the court of appeals recognized that several other courts had previously discussed a defense of copyright misuse, in analogy to the well-established patent misuse defense, and even recognized that "[a]lthough it has yet to apply the copyright misuse defense, the United States Supreme Court has given at least tacit approval of the defense". However, the court concluded that "[i]n absence of any statutory entitlement to copyright misuse defense, defense is solely equitable doctrine" and "[a]ny party seeking equitable relief must come to court with 'clean hands'". Unfortunately, this was not the case of Atari (that had falsely stated to the Copyright Office that it needed the copyrighted program it wanted to analyze for use in a litigation), so that the court did not further discuss the issue. Another relevant case for copyright misuse is Sega.[272] As observed by McManis:[273]

> "while discussing the fourth fair use factor, the court of appeals in Sega remarked that an attempt to monopolize the market by making it impossible for others to compete runs counter to the statutory purpose of promoting creative expression and cannot constitute a strong equitable basis for resisting the invocation of the fair use doctrine. This remark offers support for the conclusion that, not only was the purpose and character of Accolade's use a fair one, but Sega's use of the lockout device constituted a misuse of its copyright."

Any of the interpretations of copyright misuse reviewed by K. Judge[274] could be used to consider illegitimate this kind of licensing conditions, at least in non individually negotiated contracts (for which it is more reasonable to defer to the contractual autonomy of the parties, unless there is some clear disparity in

---

[265] Shrink-wrap licenses are software licenses which are considered accepted just because the user opens the plastic film enveloping a software package. The "virtual" equivalent is a click-wrap license, being accepted simply clicking on an "I accept" button displayed while downloading or installing a piece of software. *See* CHARLES R. MCMANIS, *The Privatization (or "Shrink-Wrapping") of American Copyright Law*, 87 California Law Review, 173--190 (1999).

[266] To discuss this issue from the point of view of contract law, the first (problematic and) preliminary issue would concern the definition of software licenses as "contracts".

[267] Antitrust "remedies" to this possibility may be relevant, but they presuppose the existence of a dominant position (in order to be qualified as abusive) and or must be equivalent to a cartel (in order to be considered equivalent to a cartel, which is probably a less likely possibility, but still worth mentioning).

[268] Consumer law is usually applicable only to final customers. And final customers do not usually have any interest in performing reverse engineering or similar activities, unless they are actually software producers: the open source model of software development make actually quite reasonable to contemporaneous nature of consumer and software producer in the very same licensee, but this issue exceeds the boundaries of the paper at hand.

[269] *See* K. JUDGE, *Rethinking Copyright Misuse*, 57 Stanford Law Review, 901--952 (2004). *See also*, for a much more synthetic review of the doctrine, JACQUELINE LIPTON, *The Law of Unintended Consequences: The Digital Millennium Copyright Act and Interoperability*, 62 Washington and Lee Law Review, 487 (2005), 540—543. The latter work appreciably relates the copyright misuse doctrine and the anti-circumvention measures of the DMCA.

[270] The possibility of finding, in civil law, a doctrine reproducing the effect of the common law (actually US) doctrine of copyright misuse is likely very low. Fortunately, the *Software Directive* explicitly pre-empts contracts forbidding reverse engineering and – in order not to empty this provision of any meaning – also general provisions (maybe in click-wrap contracts) forbidding the development of interoperable pieces of software should be considered void. See below and the second paper of this dissertation for more details.

[271] Atari Games Corp. v. Nintendo of America Inc., 975 F.2d 832.

[272] Sega Enterprises Ltd v. Accolade Inc., 977 F.2d 1510. In fact, a certain "key" was needed to unlock the code of Sega's console and it consisted "merely of 20 bytes of initialization code plus the letters S-E-G-A", which also triggered the display of Sega's trademark on the TV screen connected to Sega's console.

[273] MCMANIS, *IP Protection and Reverse Engineering.*

[274] Three main approaches are discussed in the paper: "the anticompetitive approach", "the formalistic and abuse of process approaches" and finally the "principled-guidelines approach" endorsed by the author. *See* JUDGE, *Rethinking Copyright Misuse*, 924 ff..

contracting power, which means that other tools – including contract and antitrust law – may be available to intervene).

I already mentioned that fair use could be used to allow literal copying in cases in which the implementation dictates the reproduction of a certain part of the specification. For instance, this is the case if a certain group of lines of code (or, for what matters, even a poem or another text) is used as a lockout code or the like. Here, I actually suggest doing more, in analogy with the following example: if you are a lock producer, your could make keys to open your locks containing your three-dimensional trademark (in the sense, that the trademark is part of the design of a "compatible" key for your lock). However, using trademark law to prevent independent technicians from copying your keys for end-users would risk being considered as a trademark-misuse (because you use your trademark to have a patent-like monopoly on the production of certain objects). In the same way, I suggest that there is (or, at least, should be) a finding of copyright-misuse[275] if you devised a copyrightable lockout code (like a "poem" used as "compatibility password") and then tried to use copyright law to prevent someone from realising a compatible program reproducing your lockout code.[276]

Finally, where an undertaking is dominant, competition law could be used – alternatively or complementarily to copyright misuse – to prevent it from using its intellectual property rights in ways similar to the ones I described. However, the advantage of an intellectual property doctrine like copyright misuse is that it would not require a finding of dominant position.[277]

9.1.2. Copyright and patent preemption of restrictions on reverse engineering

In the European Union, article 9 of the *Software Directive* explicitly forbids any contractual limitation to software reverse engineering for interoperability purposes, so that "[a]ny contractual provisions contrary to article 6 […] shall be null and void." The scope of this provision clearly looks very broad, encompassing not only consumers' contracts (and click-wrap and shrink-wrap licenses), but also individually contracted agreements (which is a policy choice that could be discussed and probably criticized).

In the US, the situation is far from clear. Prof. McManis,[278] for instance, is (moderately) favourable to a reading of US law accommodating for a Copyright pre-emption of anti-reverse-engineering clauses. His reasoning starts from Section 301(a) of the US Copuright Act, stating that all "rights that are equivalent to any of the exclusive rights within the general scope of copyright as specified by section 106 in works of authorship that are fixed in a tangible medium of expression and come within the subject matter of copyright as specified by sections 102 and 103 […] are governed exclusively by this title." According to the author, "The legislative history makes clear that the primary purpose of § 301 was to preempt the common-law copyright protection for unpublished works", but also to prevent from protecting works that failed to achieve Federal statutory copyright protection. McManis continues analogizing click-wrap and shrink-wrap licenses to an *ad hoc* level of protection for a given work, which is not create by state law, but which is almost equivalent to it, so that "§ 301(a) itself might preempt any effort to provide contractual protection against reverse engineering of a publicly distributed computer program, where the program would fail to achieve federal copyright protection against such reverse engineering because of the fair use provisions of § 107." In this way, the author concludes that while it is debatable whether individually negotiated contracts may or not restrict the possibility of performing reverse engineering (and I think that there may be several unintended consequences in forbidding this freedom of contract), shrink-wrap and click-wrap licenses – which are equivalent to copyright law for the licensed work – should not be able to restrict fair uses and reverse engineering (and even less to prevent the creation of interoperable works, since this would be precisely contrary to the main policy goal of the Copyright Act and probably constitute copyright misuse).

The analogy between shrink- and click-wrap licenses and state law is discussed also by other authors, with similar conclusions:

> "The reason that these contractual derogations carry such '*erga omnes* effect' - resembling an alternative legislation under control of a private party - is that such licenses are unilaterally drawn by the copyright

[275] See § *9.1.1. Licenses and copyright misuse.*

[276] Lexmark International, Inc. v. Static Control Components, Inc., 387 F.3d 522 case could provide useful arguments in this field.

[277] See the third paper for a discussion of the concept of dominant position and for a competition policy based analysis of interoperability issues in general.

[278] MCMANIS, *IP Protection and Reverse Engineering.*

> owner and their diffusion leads, in certain markets, to a situation where anybody who wants to access a certain type of content must enter into a contractual relationship with that party.
> Some commentators argue that certain types of contractual derogation to copyright law should be always void, while others ask for a case by case judgment. Robert Merges, concerned that they are not negotiated and particularly widespread in a certain market, has suggested that these derogations should be void when they resemble a kind of private legislation: '[s]tandard form software licensing contracts by virtue of their very uniformity and the immutability - in other words, non-negotiability - of their provisions, have the same generality of scope as the state legislation that is often the target of federal preemption.'"[279]

These conclusions are made even sounder by the fact that it is today possible to condition the working of a given piece of software to the acceptation of various kinds of conditions, possibly forcing the user to be connected to the Internet to scroll down a contract. Hence, it is difficult to argue that "third parties" could escape the license, thanks to the first-sale doctrine or similar principles.

Moreover, not only copyright, but also patent law could prevent some anti-reverse engineering provisions[280]. In the Bonito Boats[281] case, the Supreme Court also found that a Florida statute, prohibiting the use of a molding process to duplicate (unpatendent) boat hulls, conflicted with federal patent law and was thus invalid under the "Supremacy Clause." The Court stated:

> "In essence, the Florida law prohibits the entire public from engaging in a form of reverse engineering of a product in the public domain. This is clearly one of the rights vested in the federal patent holder, but has never been a part of state protection under the law of unfair competition or trade secrets."

Of course (and once again), click-wrap licenses are not state law; hence, it is debatable that they could be preempted as a Florida law. However, their practical effect is almost as generalised as that of the state law, hence I do share the point of view of Prof. McManis, according to which an application of patent preemption would be sensible. That is coherent with other decisions (as found by McManis), like Brullotte v. Thys Company (concerning the obligation to pay royalties beyond the expiry date of a patent). According to the author, "these cases stood for the proposition that contracts which attempt to provide protection unavailable under federal patent law are preempted."

9.2. May patent law (as currently applied to software) limit interoperability?

The bulk of this paper focuses on copyright/author's rights, since this is the main tool used to protect software innovation and because case law concerning interoperability focused on copyright violations. In fact, not only is copyright the usual tool to protect innovation in the software fields, but the great majority of so-called "software patents" concern indeed "software implemented inventions" and are actually not developed by software houses and tend to apply more to the software/hardware interaction or to software embedded in specific devices, different from normal personal computers.[282] Nevertheless, the role of patents in protecting innovation achieved through software is growing and it is appropriate to also analyse the possibility that patent law – as applied to software – may hinder interoperability.

In the US the answer is probably yes, at least potentially: it may be possible to hinder interoperability using software patents. In fact, at least for "pure software patents"[283] it is possible that interoperability itself (or a procedure needed to achieve it) may be the object of a patent. The only qualification to this is that the number of valid patents concerning communication protocols and other interoperability information is not likely to be very high. As shown by the evaluation of Microsoft protocols performed by the European Commission and already discussed, not many communication protocols and APIs are likely to be innovative enough to involve an inventive step (and, in several cases, proprietary versions of public standard do not add

---

[279] *See* OTTOLIA & WIELSCH, *Legal Aspects of Modularization and Digitalization*, text accompanying footnotes 286—292.

[280] The following view is actually grounded in the US, not only because it is based on US case law, but also because the existence and status of software patents in Europe is debatable. In any case, since in the EU patent protection is *de facto* available for some software related inventions, there are reasons to try to apply a similar general reasoning to Europe as well.

[281] Bonito Boats, Inc. v. Thunder Craft Boats, 489 U.S. 141 (1989), also quoted by MCMANIS, *IP Protection and Reverse Engineering*; SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering* and others.

[282] According to some estimates, the software industry is granted only 5% of all software patents. *See* JAMES BESSEN & ROBERT M. HUNT, *An Empirical Look at Software Patents*, 16 Journal of Economics and Management Strategy, 157--189 (2007).

[283] Shemtov suggests that even decompilation could be hindered by pure software patents. See NOAM SHEMTOV, *Rethinking Entitlements in the Context of Decompilation of Computer Programs: A Market-Based Perspective*, Working paper presented at the 3 rd Annual Workshop on the Law and Economics of Intellectual Property and Information Technology, 5-6 July 2007, Queen Mary, University of London (July, 2007).

much to the public version of the protocol, apart from some secret and undocumented changes, generating incompatibility with third party programs). Moreover – in several cases – it should be possible to "invent around the patent", at least if the claim has been kept appropriately narrow.[284] At the same time, it is quite clear that – in this context – there may be significant inefficiencies, if low quality patents are granted in the field of software:

> "The big problem is, there is often no way to circumvent naïve ideas, and there is no way whatsoever to ensure interoperability with patented protocols, formats or standards, if the patent owner refuses to licence them. […] This means that a broad enough/vague enough/trivial enough patent is a true licence to kill competitors."[285]

One of the most crucial problems in this field concerns how "abstract" patent claims are allowed to be. This issue is considered in particular by Bessen and Meurer 2008.[286] As the authors put it, "The distinguishing feature of an abstract patent claim is not that it covers a broad range of technologies, although that is often the case, but rather that it claims technologies unknown to the inventor."[287] In the field of software interfaces, I argue that granting patents covering any technique of implementation of a given specification – including techniques of implementation that the patent holder never thought about – could frequently border on the field of abstract patents. However, I do not think that this issue could be tackled at such an abstract level and – depending on the kind of technical problems that a specification addresses – it is reasonable to argue that software patents, at least in the US, may cover the entire specification (so that any implementation would be infringing). Indeed, the US Patent Office does not require source code, nor detailed flowcharts, to be attached to applications concerning software. Hence, it is very likely that a patent concerning interoperability would resemble much more an interoperability specification, than its implementation; thus independently developed implementations as well would be covered by the exclusive right.

The only good news for sustainers of interoperability is that I am not aware of similar patents covering major software technology (i.e. technologies needed in order to have general purpose systems communicating with one another). Should such a patent exist and be deemed to be enforceable (which is another significant issue in the field of software patents), intellectual property as it stands would probably be unable to guarantee that some ways to achieve interoperability do exist. Should the patent holder refuse to license his or her innovation, late comers would likely have to resort to competition policy in order to get a license at reasonable and non-discriminatory conditions, but this – to be sure – would be possible only in case of quasi-monopolistic or otherwise dominant positions.[288]

Even though the previous description concerned the US in particular, also in Europe, and despite significant existing limitations to the patentability of software "as such", it is not so easy to rule out the possibility that patents may significantly hinder interoperability. For instance, Välimäki[289] suggests that: "European copyright laws have a well-established principle that a single right owner cannot control interoperability information through copyright. Unfortunately patent law does not know such exception." In fact, I share these doubts, however I do not completely agree with the likelihood of major problems

---

[284] For instance, in the second paper, I will mention that – at the conclusion of the European Microsoft (IV) case – Microsoft licensed (for a mostly symbolic fee of 10.000 euro and under a non-disclosure agreement concerning the original version of the disclosed interface specification documents) a significant amount of interoperability information to Samba. (Samba it an open source project, which is crucial to access mixed networks from Linux clients and for workgroup servers involving various kind of operating systems in general. The slogan of Samba is quite telling: "Opening Windows to a Wider World".) That agreement did not allow Samba to use Microsoft's patented technology, which could possibly be necessary to implement these specifications. However, Samba developers are confident to be able to invent around Microsoft patents and the development of Samba seems to be flourishing, with a 4.0 version currently in alpha test and aiming at making Microsoft's Active Directory logon protocols (used by Windows 2000 and above) available to non-Windows users (see the second paper of this dissertation and http://us3.samba.org/samba/ for more information).

[285] ROBERTO DI COSMO, *Legal Tools to Protect Software: Choosing the Right One*, 4 UPGRADE (2003).

[286] *See* JAMES BESSEN & MICHAEL J. MEURER, *Patent Failure*, (James Bessen ed., Princeton University Press. 2008), ch. 9.

[287] Id., § 9.2.2.

[288] However, a dominant position is far from rare in the field of software, and could be almost likely in case a patent preventing interoperability covered a crucial technology. In fact, a software industry incumbent may easily achieve market shares of more than 80 or 90 percent. These shares are sometimes contestable, but backward compatibility is frequently a prerequisite to displace incumbents. If compatibility is precluded by a patent – and since patents last for about 20 years, that is much more that the life cycle of two or three generations of software products – a stable market share is much more likely, and so is dominance.

[289] See VÄLIMÄKI, *Software Interoperability and IP*, and VÄLIMÄKI & OKSANEN, *Patents on Compatibility Standards and Open Source*. The author suggests to use other tools (with respect to IP) to limit the use of patent law to impede interoperability, including "competition law, industry standardization bodies, and government procurement policies".

hindering interoperability, because of the absence of pure software patents in Europe, so that it is difficult to imagine a patent covering (and hence impeding) software interoperability (while it is easier to do so in the hardware field).

Here, it may be appropriate to distinguish between vertical and horizontal interoperability. In fact, where pure software patents are not admitted (and hence software '*per se*' is not patentable, and in a quite strict sense), it is difficult to envisage how a software patent could limit classical vertical interoperability. In these cases, indeed, a new piece of software must interact with another installed and patented piece of software, and interoperability would normally not require the new software to implement the patented technology. What is necessary, on average, is that the new software is just able to ask to the original one to perform some task. That is true because the patent should not protect the process of exchange of data between pieces of software – which is, to me, a quite clear case of software activity '*per se*' or 'as such'; instead, patents could cover some activities performed by a piece of software and leading to industrial results. However – in the case of horizontal interoperability, including cases in which it is important to read a common file format – it may be the case that there are problems in re-implementing the necessary features of a protected software, without violating a patent on a software related invention.

Actually, there is at least one reason to think that interoperability should not be prevented by software patents in Europe. And that is the fact that such an outcome would not only be completely incompatible with the policy of the European Commission with respect to interoperability – as outlined in the European Microsoft (IV) case. Moreover, it would also contradict the votes of the European Parliament concerning the proposed (and failed) Directive concerning software implemented inventions (the legislative history of which I will briefly touch in the second paper of this dissertation). In particular, discussing the proposed Directive, the Parliament included the following article 6a:

> Member States shall ensure that, wherever the use of a patented technique is needed for a significant purpose such as ensuring conversion of the conventions used in two different computer systems or networks so as to allow communication and exchange of data content between them, such use is not considered to be a patent infringement.[290]

A much less revolutionary "decompilation exception" had already been inserted in previous draft of the Directive, however, this exception simply assured the possibility of performing reverse engineering of patented interfaces, without clearly granting a right to re-implement them, in case there were apparently no way to "invent around" the existing patent. According to Chapin,[291] the Parliament proposed exception

> "would not render a software patent useless; rather, it would merely ensure the ability of new software to communicate freely with the patented software or to allow for new standards to be created that could communicate with and convert output from previously patented standards."

However, it is evident that not everybody agreed. In fact, risks for interoperability were one of the main arguments of those who opposed the draft Directive. And the fact that the Parliament insisted on a clear interoperability exception (not limited to the possibility of decompilation, but also granting a kind of "right to re-implement") in the field of software implemented inventions was probably one of the points that – at the end of the day – also convinced the proponents of the Directive that a confirmation of the *status quo* (i.e. the absence of any clear rule) was preferable. So, with both sides being unable to obtain what they wanted, the whole proposal was turned down in July 2005 (when the Council of Ministers resubmitted it to the Parliament having removed article 6). In fact, today there is a high degree of uncertainty concerning the validity and the degree of enforceability of software patents in Europe (which are more or less generously granted by the European Patent Office),[292] and some dominant incumbents could arguably prefer such a

---

[290] See Consolidated version of the amended directive "on the patentability of computer-implemented inventions" for which the European Parliament voted on 2003-09-24 (available at http://eupat.ffii.org/papers/europarl0309/index.en.html ; last visited August 3, 2008). About the wording of this article (and its inadequacies), see also MARCO RICOLFI, *Tutela della concorrenza, proprietà intellettuale e TRIPs*, in Antitrust e globalizzazione 141—172 (2004), pp. 166-169.

[291] MICHAEL CHAPIN, *Sharing the Interoperability Ball on the Software Patent Playground*, 14 Boston University Journal of Science and Technology Law, 220 (2008), p. 235.

[292] That the European Patent Office has granted a significant number of patents that could be reasonably labelled "software patents" is recognized by several commentators. *See* VÄLIMÄKI & OKSANEN, *Patents on Compatibility Standards and Open Source*, p. 402. However, it is difficult to gather credible data, since there is clearly no "software patents" category in the registries of the office (since, in principle, only "software implemented inventions" may be protected). A number of EPO-granted "software patents" between 20.000 and 30.000 has been extensively quoted starting from 1995, in the context of the debate around a potential Software Patents Directive, but I confess that I have never been able to find the original source of this estimate.

situation of uncertainty to a clearer legislative setting, where software patents are *de facto* allowed, but with a clear-cut interoperability exception.

To conclude, the European field of software patents is significantly blurred and would deserve a specific research in itself,[293] in particular because of the significant misalignment between the legal setting "on paper" and the actual working of the EPO. Unfortunately, such an analysis would lie largely outside the scope of this paper. However, there are some clear confirmations of the fact that obstacles to interoperability – also, and probably in particular, in the field of horizontal interoperability – are less pronounced in Europe than in the US. That is confirmed by the fact that several Linux distributions (and other open source operating systems) have a specific CD or DVD, containing pieces of software, usually needed in order to access to some specific multimedia file formats, the distribution and use of which is excluded in the US,[294] also because it could constitute a patent infringement.

9.2.1. If patent can be used to hinder interoperability, is it a good idea to do so?

Surely in the US, and – in some limited cases – possibly also in the EU, patents may be used to prevent the reimplementation of some (patented) specifications. However, fortunately for the software industry, several commercial software houses already realized that it might not be wise to collect royalties for the use of APIs, communication protocols, file format and similar software objects, the value of which is greatly increased by widespread interoperability. Indeed, in several cases, that happened without any particular regulatory pressure (even though antitrust policy could also have urged in this direction some software houses that would not have done so otherwise). In particular, several firms promised not to assert their patents against the open source community. IBM was one of the first companies to allow for the use of 500 of its patents in January 2005;[295] SUN Microsystems also granted the possibility of using some 1600 of its patents.[296] Novell also promised to fight back, using its patent portfolio, against any attack to the Linux Kernel some other open source software.[297]

In 2005, the Open Invention Network was created, using investments from IBM, NEC, Novell, Philips, Red Hat[298] and Sony, with the goal of "using patents to create a collaborative environment",[299] following the principle that Patents owned by OIN would be "available royalty-free to any company, institution or individual that agrees not to assert its patents against the Linux System".[300] In August 2007, Google became a licensee of the project.[301] A specific project, Patent Commons, has even been launched to map all similar contributions to the "patent common". In addition to the firms already mentioned, listed contributors include Computer Associates, Ericsson, Nokia, HP, Oracle and others.[302] Even Microsoft – frequently described as "The Enemy" of open source – launched (and recently extended) its Open Specification

---

[293] For some thoughts on this topic, see CHAPIN, *Sharing the Interoperability Ball*, pp. 234—236, according to whom "the status of software patents in general, and the status of interoperability between patented computer programs and application software inventions, remain in a state of uncertainty within Europe".

[294] In fact, even the servers to distribute them were special mirrors outside the US (see http://non-us.debian.org/README.non-US for more information).

[295] See ECT News Business Desk, *IBM Boosts Open Source with Patent Promise*, on TechNewsWorld.com (available at http://www.technewsworld.com/story/35584.html; last visited July 28, 2008).

[296] See Sun's press release "Sun Grants Global Open Source Community Access to More than 1,600 Patents", SANTA CLARA, Calif. - January 25, 2005 (available at http://www.sun.com/smi/Press/sunflash/2005-01/sunflash.20050125.2.xml; last visited July 28, 2008).

[297] "Novell will use its patent portfolio to protect itself against claims made against the Linux kernel or open source programs included in Novell's offerings, as dictated by the actions of others. […]Novell is prepared to use our patents, which are highly relevant in today's marketplace, to defend against those who might assert patents against open source products marketed, sold or supported by Novell." See Novell's Patent Policy (available at http://www.novell.com/company/policies/patent/; last visited July 28, 2008).

[298] About the patent policy of Red Hat, see *Red Hat, Inc. Statement of Position and Our Promise on Software Patents* (available at http://www.redhat.com/legal/patent_policy.html): "to the extent any party exercises a Patent Right with respect to Open Source/Free Software which reads on any claim of any patent held by Red Hat, Red Hat agrees to refrain from enforcing the infringed patent against such party for such exercise." See also Greg DeKoenigsberg, *The Red Hat Patent Promise: Encouraging Innovation*, Red Hat Magazine, Issue 1, November 2004 (available at http://www.redhat.com/magazine/001nov04/features/patents/) (both last visited July 27, 2008).

[299] See http://www.openinventionnetwork.com/.

[300] http://www.openinventionnetwork.com/.

[301] See official OIN press release (http://www.openinventionnetwork.com/press_release08_06_07.php).

[302] See http://www.patentcommons.org/.

Promise initiative and other covenants not to assert its intellectual property rights in certain specific domains (see below). For this reason, some commentators[303] optimistically argue that "[e]ncouraging such company practices might be the best option for a government if it considers patent royalties on compatibility standards a policy problem."

Again about Microsoft, in its very recent "Interoperability Principles" document, about "Open Connections, Standards Support, Data Portability"[304], the software house opted – in several domains – for an "open access" policy – without using the typical non-disclosure-agreements and without imposing access fees. Indeed, "Microsoft will not require developers to obtain a license, or to pay a royalty or other fee, to have access to all this information". This seems to imply (implicitly, of course) that, according to Microsoft too, interoperability information protected only by copyright is normally freely re-implementable by third parties. However, when software patents are involved, there is a clear distinction between access and use of interoperability information. In fact, Microsoft also warns developers that some patent-protected interfaces could require the payment of (reasonable and non-discriminatory - RAND) royalties.[305] It should also be noticed that, in the same document, it was originally stated that "Microsoft will covenant not to sue open source developers for development and *non-commercial* distribution of implementations of these Open Protocols" (emphasis added)[306]. However, while this article was being finished, Microsoft significantly updated its Open Specification Promise,[307] and Sam Ramji, Director of Microsoft's Open Source Software Lab, publicly stated that:

> "Microsoft is putting a wide range of protocols that were formerly in the Communications Protocol Program under the Open Specification Promise (OSP). This guarantees their freedom from any patent claims from Microsoft now or in the future, and includes both Microsoft-developed and industry-developed protocols.
> We have established a clarification to the OSP that guarantees developer rights to build software of any kind and for any purpose using these specifications, including commercial use."[308]

In a way, Microsoft seems to have decided to behave – limited to some specific protocols – more or less like a standard-setting authority. Of course, as a not very open standard setting authority, where freely accessible specifications and RAND royalties are perceived as sufficient.[309]

Overall, I do not think that these developments show that incumbents do not want to use their patents in order to foreclose competition and hinder interoperability. It just means that some firms (like IBM) – which are not leaders in mass software markets – may like to use some of their patents in order to stimulate an efficient decentralised production of software, which is complementary to their software and hardware (!) products. Other firms, like Microsoft, additionally have to worry about the fact of being quasi-monopolists in several markets and may adopt a not-completely-closed policy, in order to reduce the pressure from the

---

[303] VÄLIMÄKI & OKSANEN, *Patents on Compatibility Standards and Open Source*.

[304] See Microsoft's *Interoperability Principles – Open Connections, Standards Support, Data Portability*, published: February 21, 2008 (text retrieved on February 25, 2008) (available at http://www.microsoft.com/interop/principles/default.mspx; last visited July 28, 2008).

[305] See Microsoft's *Interoperability Principles* (supra note 304): "RAND Patent Terms. Some of Microsoft's Open Protocols are covered by patents. Microsoft will indicate on its website which protocols are covered by Microsoft patents and will license all of these patents on reasonable and non-discriminatory terms, at low royalty rates. To assist developers in clearly understanding whether or not Microsoft patents may apply to any of the protocols, Microsoft will make available a list of the specific Microsoft patents and patent applications that cover each protocol. We will make this list available once for each release of a high-volume product that includes Open Protocols. Microsoft will not assert patents on any Open Protocol unless those patents appear on that list. Third parties do not need licenses to any Microsoft patents to call these Open APIs".

[306] See Microsoft's *Interoperability Principles* (supra note 304).

[307] See *Microsoft Open Specification Promise*, published: September 12, 2006 and last updated: July 25, 2008 (available at http://www.microsoft.com/interop/osp/default.mspx; last visited July 28, 2008).

[308] Sam Ramji, Director of Microsoft's Open Source Software Lab, quoted by Matt Asay, *Microsoft opens up its Open Specification Promise*, July 25, 2008 (available at http://news.cnet.com/8301-13505_3-10000124-16.html; last visited July 28, 2008). See also *Microsoft Open Specification Promise*, *supra note* 307.

[309] For more stringent requirements for standard setting authorities, see, for instance, the "European Interoperability Framework" document – *European Interoperability Framework for pan-European eGovernment Services* (available at http://europa.eu.int/idabc/3761; last visited August 5, 2008) – drafted in the context of the European Union's software procurement rules. In this document, standards are defined as "open" based on three criteria: the standard has to be adopted maintained by a non-profit organization and on the basis of an open decision-making procedure; the standard specification documents have to be available freely or at a nominal charge; the intellectual property of (parts of) the standard must be made irrevocably available on a royalty free basis. After the recent modification to its Open Specification Promise, this last condition seems to be respected by Microsoft, at least for the covered protocols and toward open source projects.

antitrust authority. However – as the recent Microsft-Novell agreement,[310] where "[t]he two companies also announced an agreement to provide each other's customers with patent coverage for their respective products" – all these developments are consistent with the fact that software patents are used (both in a pro- and anti-competitive way) almost always as big bundles of patents. In fact, the basic idea of a patent agreement between major firms in the software market is: "I do not sue you if you violate my n-hundred patents and you don't sue me in the corresponding case." In this way, a single software patent is basically worthless as a stimulus for innovation, while patent-thickets become powerful barriers to entry in the software industry (especially for smaller players). Press releases typically concern n-hundred patents used in a certain way or violated.[311] That is coherent with the fact that developers frequently unconsciously write down code that could violate a certain software patent, also because the standards of quality to be granted a software patent are not always rigorous: at the end of the day, the only way to write code without spending an excessive amount of resources in auditing code against potential involuntary violations of some other firm's patents is to reciprocally "clear" all patent issues. That is quite telling about the transaction costs generated by software patents and about the barriers to entry faced by firms outside this network of crossed promises not-to-sue.

As a final note concerning software patents, it may be appropriate to notice that – in particular because of the frequently shaky nature of claims in this field – it may not be a good idea to try to enforce a software patent precisely in a case where software interoperability is at hand. Indeed, a recent US precedent gives to patent holders additional issues to think about. In fact, it has been observed[312] that in eBay v. MercExchange:[313]

> "[t]he Court made clear that there should not be a general rule in patent cases that a finding of infringement and validity guarantees a permanent injunction against the infringer. Rather, the Court upheld the traditional four-factor framework that requires a plaintiff to show:
> '(1) that it has suffered an irreparable injury; (2) that remedies available at law, such as monetary damages, are inadequate to compensate for that injury; (3) that, considering the balance of hardships between the plaintiff and defendant, a remedy in equity is warranted; and (4) that the public interest would not be disserved by a permanent injunction.'
> In so holding, the Court effectively gave those seeking access to patented technology significantly more leverage in licensing negotiations. Addressing possible exceptions to the general rule, Justice Kennedy's concurrence also recognized that '[w]hen the patented invention is but a small component of the product the companies seek to produce and the threat of an injunction is employed simply for undue leverage in negotiations, legal damages may well be sufficient to compensate for the infringement and an injunction may not serve the public interest.' Software interoperability seems to fit nicely into this category since interoperability functions of a software invention should only comprise a very small amount of the total product."

Hence, given the importance of preliminary rulings in terms of credibility of patent strategies – which are frequently based on threats, based on supposedly valid patent thickets – firms should be very careful in trying to enforce these shaky proprietary titles, in one of the fields where they may be weaker. This is probably another reason why it is very difficult to find cases, concerning patents and software interoperability, where the outcome of the case has not been a private settlement.

---

[310] See press release: "Microsoft and Novell Announce Broad Collaboration on Windows and Linux Interoperability and Support", REDMOND, Wash., and WALTHAM, Mass. — Nov. 2, 2006. (Available at http://www.microsoft.com/presspass/press/2006/nov06/11-02MSNovellPR.mspx. Last visited July, 27, 2008.) See, in particular, the "Patent Cooperation Agreement - Microsoft & Novell Interoperability Collaboration" page on Microsoft's website, November 2, 2006 (updated July 5, 2007) (available at http://www.microsoft.com/interop/msnovellcollab/patent_agreement.mspx last visited, July 27, 2008): "Microsoft, on behalf of itself and its Subsidiaries (collectively "Microsoft"), hereby covenants not to sue Novell's Customers and Novell's Subsidiaries' Customers for infringement under Covered Patents of Microsoft…".

[311] For example, Microsoft recently declared that the Linux system would violate more than 250 Microsoft's patent. The specific patents number remains unknown and the legal threat seems to rely on the "number" of patents, more than on their nature and importance. *See* Roger Parloff, *Microsoft takes on the free world*, FORTUNE Magazine, May 14, 2007, http://money.cnn.com/magazines/fortune/fortune_archive/2007/05/28/100033867/ (last visited July 21, 2008).

[312] *See* CHAPIN, *Sharing the Interoperability Ball*, pp. 243—244.

[313] Case eBay v. MercExchange, 547 U.S. 388 (2006).

9.3. A possible economic criticism (IP as a tool enabling desirable business models)

In this paper I essentially argued that preventing free riding on software development expenditures (while allowing it on ideas and – as long as one is able to access them through decompilation – API specifications) is sufficient to stimulate innovation in software markets. I largely based my conclusion on legal reasoning, but I reinforced the same conclusion on the basis of economic elements. In a few words, I argued that we should avoid the risk of defining property rights on objects (pieces of interface information) which are valuable more because they are used (by producers of complementary products and or users) and not so much because of especially creative, innovative or productive investments of the original developer. In other words, I think that there are good reasons to believe that software interfaces would be created in any case, even if they were not protected by any kind of intellectual property right. However, a relevant economic criticism to my approach is that the "cost" of this (narrow) definition of intellectual property rights could be that of preventing (not the creation of software interfaces, but) some marketing strategies that require significant market power and control over interfaces. Moreover, the prevention of these marketing strategies may be problematic, because they may (in some cases) be socially beneficial: in particular, I am referring to some price-discrimination and some two-sided strategies.

Price discrimination strategies may allow the financing of investments in research and development by "taxing more" those users with higher willingness to pay (e.g. professional users instead of students): in some cases, the net effect of this strategy may be that of collecting more resources to innovate, creating a lower dead weight loss for the society.[314] It is theoretically possible[315] that students' discounts are actually made possible by the existence of a high mark up on (for instance) professional users and that – if a single price must be chosen – it is high enough to exclude several students from the market. Unfortunately, competition may prevent firms from practicing price discrimination (because competitors may try to undercut the price offered to high willingness to pay users), and that may happen even in cases in which price discrimination would have been beneficial. Similarly, two-sided strategies[316] may allow for the squeezing of one side of the market (for example users) in order to subsidize the other (for example developers of complements) and this could be "socially" useful (apart from being lucrative), if one side is more apt to generate network effects (or needed to reach a critical mass of complementary goods, users and so on). Clearly this strategy would be undermined, if someone else could undercut the leader on (in my example) the users' side, by producing a compatible platform enjoying the indirect network effects generated by producers of complements (and enjoyed by users).

To summarize, in these and similar cases, a certain player in a quasi-monopolistic condition may have such a high degree of control of a given market that it becomes in its interest to avoid some market failures and internalize some externalities concerning this market. And that simply because having a bigger and better market allows it to extract more surplus from it. So, it is theoretically possible that APIs (and other interfaces) are not very costly in themselves to develop, and hence there is no real reason to protect them with IPR in order to stimulate their initial production. However, what may be useful is to control them in order to implement various pricing strategies, and since these pricing strategies could be beneficial for users (as demonstrated by several papers),[317] it may be that a monopoly on APIs is beneficial, even if the reason is not that we need to create incentives for their production. For instance, what really needs to be incentivized may be an internalisation policy between two sides of a market. In other words, if the control on APIs could make possible two-sided strategies that may be beneficial to the public, this may be one of a few cases in which there are in fact *ex post* (with respect to the moment of creation) justifications for intellectual property rights.

I admit that this can be the case. Theoretically, it is possible and I will not try to argue that it is not. What I can reply to this line of criticism is that my "educated guess" is that distributing only the binary code of one's

---

[314] Price-discrimination strategies may be socially beneficial in any market with market power (because it may increase the quantity actually produced), but in IP markets there is the additional advantage that some of the additional "extracted profits" may be used to finance innovation in a relatively efficient way, because these investments are paid more by consumers with a less elastic demand.

[315] *See* JEAN TIROLE, *The Theory of Industrial Organization*, (MIT Press, Cambridge: Mass. 1988).

[316] *See* ROCHET & TIROLE, *Two-Sided Markets: A Progress Report*, (or ROCHET & TIROLE, *Two-Sided Markets: An Overview*). See also J. C. ROCHET & J. TIROLE, *Platform Competition in Two-Sided Markets*, 1 Journal of the European Economic Association, 990--1029 (2003); ROCHET & TIROLE, *Platform Competition in Two-Sided Markets*; MARK ARMSTRONG, *Competition in Two-Sided Markets*, 37 RAND Journal of Economics, 668--691 (2006).

[317] See, in particular, *supra notes* 315 and 316.

platform (i.e. coupling copyright and trade secret) is a more than sufficient tool to keep enough lead time and technical barriers to entry. In most cases, this will be sufficient to implement two-sided strategies. Indeed, the literature arguing that operating systems are indeed two-sided platforms based this observation on the real world examples, where vertical and horizontal interoperability are both possible, even if not so easily achievable. In other words, intellectual property rights cannot be used to absolutely exclude any players, but platform controllers are still able to favour the ones, which two-sided (or price discrimination) strategy suggests to subsidize.

Notice that, in this paper, I am talking about late comers creating interoperable systems through self-help: typically, reverse engineering followed by an independent reimplementation. Instead, I am more willing to concede that – in some cases – the effect of an antitrust authority mandating interoperability (especially if in the form of free disclosure of API specifications or even of the source code of implementations) could be to lower social welfare,[318] if the potentially positive effects of price discrimination and two-sided strategies are completely neglected. However, this scenario derives from the fact that mandated interoperability drops the cost of reverse engineering to zero, which is a significant alteration of the "normal" scenario, in which a competitor has to "pay" in terms of decompilation efforts to access API specifications.[319,320] Finally, also notice that – even if a platform controller is in the best position to manage positive externalities generated "around" its system – there are several cases in which the platform controller's strategies are not socially beneficial at all. (Here I refer to the Farrell and Weiser paradigm of the Internalisation of Complementary Efficiencies and – in particular – to its exceptions.)[321] It is also because of this risk of abuses (that I will discuss in the third paper of this dissertation) that I argue that the level of control over interoperability that could be granted by the "technology copyright" is normally more than sufficient.

10. Conclusions

The definition of the legal status of APIs and CPs comes from a balancing exercise concerning incentives to innovate – on one side – and constraints to subsequent derivative and complementary innovation – on the other. Despite the possibility of addressing some of these problems *ex post*, using competition policy, I think that it is important to address this trade-off also in the initial stage consisting in the definition of intellectual property rights. In fact, as is has been made clear in the Magill and IMS-Health antitrust European cases, poorly defined intellectual property rights are likely to push competition policy in the direction of an excess of intervention or of the definition of blurred rules.[322] At the same time, I think that a complete solution of interoperability-related problems cannot completely rely on IP law and cannot prescind from the role of competition policy. Indeed, antitrust activity addresses the use of market power, which may be higher or lower depending on the configuration of intellectual property rights. Hence, if competition policy is very effective in addressing abuses likely to hinder innovation, then intellectual property rights can be broader and stronger. In addition, technological conditions can allow a combination of contracts and trade secret to complement or partially substitute IP rights: in these cases competition policy is the better candidate, at least in the short run, to recreate an appropriate balance between the incentives to innovate of the first comer and the possibility of subsequent innovation by other players. However – in certain fields – the social costs of

[318] In this case – since interoperability is mandated and disclosure free – the barrier to entry represented by the cost of decompilation is no more at work.

[319] I am not aware of any two-sided model considering the cost of reverse engineering or other barriers to achieve interoperability. My preliminary guess is that, for certain (low) costs of reverse engineering, intellectual property rights on APIs could be needed, to efficiently implement two-sided policies. However, I already explained why do not think that the real world has very low reverse engineering costs (at least for the moment; moreover, better techniques of decompilation would be balanced by more complex systems and "DRM-like" technological protections). In fact, when the cost of decompilation rises, it could become easier to have a sufficient buffer to put in place two-sided policies (this scenario is likely to be the relevant one for the real world). In these cases the problem would be: when should we mandate disclosure (and indirectly compatibility)? A clear cut answer to this second question is likely to be impossible, as demonstrated by several papers about compatibility in two-sided markets (ROCHET & TIROLE, *Platform Competition in Two-Sided Markets*; ARMSTRONG, *Competition in Two-Sided Markets*). What is clear is that – not knowing a definite general answer – one should be very careful.

[320] Indeed, the interpretation of intellectual property proposed in the paper at hand – but also in the third paper of this dissertation – is very far from implying that the cost of entering the market with a product which is complementary with that of the incumbent is zero. This cost depends not only on the cost of reverse engineering, but also on the cost of a new implementation. (Moreover, at least in the US, innovative interface specifications could, in principle, be patent protected.)

[321] See FARRELL & WEISER, *Modularity, Vertical Integration, and Open Access*.

[322] See also DREXL, *IMS Health and Trinko*.

using antitrust are prohibitive; there, a careful attribution of IP rights can be a more efficient instrument to balance incentives to innovate and the related costs.

In this setting, I think that the analysis performed in this paper allows for the drawing of at least some preliminary conclusions concerning the optimal scope of intellectual property law in the fields, which are crucial for interoperability.

First of all, it is clear to me that APIs and CPs implementations are protected by copyright, as any other piece of software. This is fine and economically sound: copyright protects first comers against complete free riding on their investments in writing source code.[323] However, it also clear that this protection is relatively thin: it certainly covers literal copying and mechanical modifications/translations, but it surely does not extend to ideas and methods. Hence, abstract interface specifications are not copyright protected, no more than the contents of a math book, even thought the book itself – as an interface specification document/manual – is protected against copying.

Moreover, in the majority of relevant cases – at least as far as the practice of the last years shows – interface specifications are usually not very "innovative" in themselves and their strategic value is mostly created by the decisions of users and producers of complementary goods. Hence, also patent law is not likely to represent a major obstacle to software interoperability. And that is even truer in Europe, where the law prevents the patenting of software "as such" and hence makes it quite difficult to claim patent protection for software tools that are aimed at allowing the exchange of data between pieces of software, and hence not likely to have a specific industrial application. That having been said, a patent-like protection for software interfaces could potentially disrupt the careful balancing between incentives to innovate and possibility of creating interoperable products as I described in this paper. Hence, legislators are advised to maintain copyright and trade secret as the main tools used to protect software innovation, thus avoiding recurring to patent in a significant way. (Actually, my suggestion is to avoid software patents completely, but my analysis was admittedly focused on the field of software interfaces, so I cannot exclude that, in principle, software patents could be necessary to protect different kinds of software innovations.)

The crucial points of the paper, that represent its main contribution to the existing literature concerning interoperability, are the constant reference to the specification/implementation dichotomy and the clear distinction between an access phase and a reimplementation phase. I already discussed at large the consequences of the first distinction. But also the second distinction is critical, as only the access phase needs – in order to be insulated from copyright liability – to enjoy a precise copyright exception (statutorily provided or coming from fair use analysis), which is typically conditional on the absence of a significant negative impact of this access to the market for (or value of) the original program. In that way, one can avoid the necessity of recurring to fair use or other "rules of reasons", which could be tempting for economists, but which could lead to excessive legal uncertainty. In fact, failing to distinguish between access to the interoperability specification and its reimplementation would require an extensive recourse to fair use and it would either suggest to discriminate between vertical and horizontal interoperability (ad suggested by Weiser) or to allow both vertical and horizontal interoperability, but adopting a blurred fair use analysis (as in the aforementioned case, Sega v. Connectix). Instead, the interpretation of copyright that I propose, not discriminating between horizontal and vertical access, is not only coherent with both the prevalent US case law and the EU legislation (as interpreted by the Commission), but it is also compatible with the sketched economic model of software markets I proposed in the *General Introduction*. In such a model, copyright is an appropriate tool to impede free riding on up-front sunk costs of first comers, but it also embeds appropriate correctives, devised in such a way not to increase development costs of late comers.

Certainly, from a static point of view it is inefficient to develop the implementation of the same specification twice, hence also the solution I propose entails inefficiencies. However, this small inefficiency – coupled with the other waste resulting from the need to perform software reverse engineering to achieve interoperability specifications – may prevent pure free riding from late comers, without allowing excessive and persistent market power into the hands of the first comer.[324] Moreover, and luckily enough, the protection of

[323] There are some decisions, like the Lexmark (387 F.3d 522) one, arguing that some pieces of software are not protected at all because of a broad interpretation of merger and '*scenes a faire*' doctrines: I think that this approach – even if yielding reasonable results in this case – is likely to set weak precedents, unclear legal standards and – despite the fact that the decision I quoted is clearly pro-interoperability, as my paper – it is likely to do more harm than good to interoperability in general.

[324] Notice also that the actual existence of this inefficiency could be prevented by the licensing of interoperability information by the original developer: hence, in several cases where a contractual solution is possible, reverse engineering may just be a threat, increasing the likelihood of a bargain among parties.

implementations and the possibility of free riding on specifications results also from basic standard copyright rules: in particular, the "originality" requirement and the idea/expression dichotomy. In fact, in March 1993, Prof. Miller argued on the Harvard Law Review[325] that – more than fourteen years after their deliberation – CONTU's recommendations were still looking essentially correct and that the copyright regime was still flexible enough to address the various concerns about its aptitude to deal with computer programs. According to Miller:

> "Several key points underlie this conclusion. First, CONTU's reasoning accords with basic copyright law principles […]. Second, the copyright decisions spawned in the years since CONTU are yielding an increasingly sophisticated, coherent, and predictable software protection regime. Third, copyright principles are flexible enough that it is not necessary to fabricate an entirely different legal regime, sometimes referred to as *sui generis* protection, for effective regulation. […] Some ambiguity in the short term seems a small price to pay to assure the flexibility and discretion that our judges need to develop a sound software jurisprudence."[326]

In fact, it seems to me that it is still meaningful to ask oneself if there's "anything new since CONTU". In the meantime, network of judicial decisions (in particular in the US) and scholarly articles adapted traditional copyright law to software. Already at the time of Miller's article, it was evident that copyright's flexibility and usefulness had played a significant role in allowing the development of the software industry. Today, however, there are even more reasons to think so. In fact, copyright – probably surpassing Miller's expectations – demonstrated itself to be especially appropriate as a tool to protect software also because it allowed – with very low transaction costs – the development of the open source model of software development. In that model, some incentives to innovate are sacrificed in order to avoid the costs of secrecy and the cost of "developing around" existing code (risking to reinvent the wheel quite often and forgetting the opportunity of improving existing code, eliminating bugs and increasing its efficiency).

In other words, copyright and trade secret (as long as reverse engineering is free) provide an extraordinary pro-competitive environment for software innovation, being able to accommodate both the traditional model of software development – that has been crucial in transforming informatics into a daily productivity and entertainment tool for both firms and general customers – and the new and promising open source model. Legislators and scholars should be aware of that and – what is more important – should become aware of the reasons for which the current system is actually working. These reasons rely on copyright and on the circumscribed, but significant, possibility of free riding on technical ideas that it entails (positive externalities or technological spillovers, if you like). For the same reasons, policymakers should also avoid – if possible – to excessively disrupt the current system, as may happen by offering excessive protection to ideas and algorithms through software patents (especially if easily warranted) or through an excessive expansions of copyright toward the protection of ideas.

All that having been said, I think that the conclusions of this paper should derive more immediately from a well-designed copyright law. Hence, a clear-cut statement that interface specifications are never protected by copyright – not even as part of the internal structure/form of a software program – would be very useful in order to eliminate some residual legal uncertainty in software markets.

11. Main open problems (left for the second and third papers)

This paper is part of a broader research project. Hence, in this very final paragraph, I recall and summarize some open problems (already sketched in the *General Introduction*), which will be addressed in the second and third paper composing the project. Since several of these problems are quite broad in scope, each of the following papers will focus on some specific open issues.

11.1. Limitations on access and decompilation – Second paper

A first problem is related to the actual possibility of accessing the information contained in software object code: this is a technical problem, but it is also strictly related to legal (and also institutional) issues. In fact, case-law (especially in the US) and legal norms (especially in the EU) concerning reverse engineering have been decided or drafted in a reality in which the free-software/open-source model of software development did not yet exist or, at least, was still not considered as relevant. In the new environment of

[325] MILLER, *Is Anything New Since CONTU?*.
[326] Id., pp. 980-981 and 1073.

software development, where open source actors are important (and in some cases the main or more "dangerous" competitors of dominant traditional software houses), it is imperative to take into account the compatibility between this model of software development and norms concerning reverse engineering.

Even if we accept – as I basically argued in this first paper – that it should normally be legitimate (in terms of copyright infringement) to replicate the parts of code which are needed to achieve interoperability (both in the vertical and horizontal sense, i.e. to create complementary or substitute products), there may be legal obstacles hindering the access to this information. In fact, there are two major obstacles to decompilation: (1) limitations concerning decompilation (software reverse engineering), both with respect to the definition of the "information needed to achieve interoperability" (a problem according to some US decisions and surely in EU, according to Art. 6 of the *Software Directive*) and concerning the disclosure of the results of the decompilation activity (that – I will argue – should be defined in a way which ensures compatibility – at least – between the open source model of software development and legitimate decompilation); (2) in particular (but not only) in the US, limitations concerning the trafficking in technological measures, which may be used not only to achieve interoperability, but also to unlock DRM systems (the DMCA contains a specific exception for interoperability, but this poses legal problems similar to the one encountered in the EU and – in general – increases the cost of decompilation which is already a dramatically costly activity).

Concerning in particular point (1), since the more credible threats to the dominant actors in several software markets are today open source projects, it is particularly worrying that FLOSS[327] projects could be the ones which may find more obstacles to accessing and effectively using interoperability information in a way which is compatible with their model of software development. In fact, as antitrust cases involving Microsoft demonstrated on both sides of the Atlantic Ocean, antitrust bodies found it appropriate to impose several kinds of disclosure obligations on Microsoft, but – at the same time – intellectual property law is (partially) hindering decompilation. This situation shows some incoherence, since decompilation could be a self-help tool for Microsoft's competitors, and making recourse to this tool easier may reduce the need for reliance on courts and antitrust authorities.

These problems will be touched on in the second paper.[328] About the European situation, it is interesting that in the recent law concerning interoperability between DRM systems, the French government – in particular in the first version of the reform, approved by the lower chamber of the Parliament – took quite explicitly into account the compatibility of the models of disclosure and interoperability with the open source model of software development.[329] This may be a signal of an increasing awareness of legislators with respect to the specificities of open source software development and to the need of accommodate these specificities in the digital copyright paradigm.

11.2. Interoperability and competition policy – Third paper

No matter how permissive rules concerning reverse engineering are, decompilation may be too costly and too difficult not because of legal obstacles, but simply for technical reasons, and this is frequently the case. Hence, there may be cases in which it would be better, from the point of view of social welfare, to not only allow free copying of principles embedded in APIs specification, but also even to force their disclosure. (These issues usually arise when a dominant firm controls a software platform and decides to exercise a high level of control – maybe up to complete integration – on complementary products. This usually happens with respect to complementary platforms or middleware – e.g. browsers, media-players or server operating systems with respect to client ones – because these pieces of software are both technical complements and potential strategic substitutes for the platform itself).

This kind of antitrust problems will not be addressed in the first two papers. Nevertheless, especially in the field of software and communication technologies, intellectual property law and competition law share the same goal: enhancing social welfare through a high level of innovation. The same can be said about different jurisdictions devising incentive schemes and constraints to shape the action of software developers:

---

[327] FLOSS is the acronym of Free/Libre Open Source Software: this definition stresses that the meaning of "free" is the one of the French word "libre" and not equivalent to "gratis".

[328] Depending on the scope of the analysis, it could also be appropriate to move to the second paper the discussion concerning preemption of clauses banning reverse engineering and similar contractual limits to decompilation, which I already briefly touched in this first paper.

[329] See YVES GAUBIAC, *Interopérabilité et Droit de Propriété Intellectuelle (with en. translation: Interoperability in Intellectual Property Law)*, 211 Revu Internationale du Droit d'Auteur, 91--139 (2007).

the United States, European Union and Japan are all targeting a higher level of innovation in the field of software and information technologies.[330]

The fact that intellectual property law, competition law and different legislators propose different (and sometime antithetic) solutions to reach the same goal highlights the number of trade-offs that should be considered when dealing with innovation. Granting a certain degree of monopoly power may be a (relatively) efficient way to provide incentives to innovate; constraining excessive exercises of market power may be needed to maintain a high degree of innovation; giving larger incentives to pioneer-innovators can reduce the level of incentives for follower-innovators; etc.

The third paper will be devoted to these kinds of issues and – since this is an already widely debated issue – it will focus on two specific points. First of all, I will propose an interpretation of these problems as technological tying[331] of the pieces of software implementing the APIs and CPs (into the operating system), so that it is possible to offer to dominant firms an alternative between disclosure and unbundling of these elements from their dominant products. In such a way the "subsidization of competitors"[332] through disclosure is not made mandatory, but it is also possible to maintain, on an equal competitive footing, all the complementary products of a given dominant software platform. Secondly, I will analyze the more relevant point (from the point of view of interoperability policy) of the verdict (concerning the Microsoft Case) that the European Court of First Instance delivered the 17th of September 2007.

Clearly, the problems addressed in this part of the thesis would be the ones related to the application of the so called "essential facility doctrine"[333] in the field of IP (including trade secret), leaving aside the literature concerning regulated standard-setting activity. If needed to access a network platform, API's and CP's can be considered as *de facto* standards, but my analysis would be related to the minimal conditions needed in order to have a decentralised (market-based) system (and not to the optimal conditions for a centralised standard setting activity in the field of software platforms). In other words, I will take into account the possibility that antitrust authorities mandate the existence of a market, but not the existence of a regulating authority in this field.

[330] EU (Commission Guidelines on the application of article 81 of the EC Treaty to technology transfer agreements 2004/C 101/02 about Commission Regulation No. 772/2004): "both bodies of law share the same basic objective of promoting consumer welfare and an efficient allocation of resources". US (Antitrust Guidelines for the Licensing of Intellectual Property issued by the Department of Justice and the Federal Trade Commission in April 1995): "The intellectual property laws and the antitrust laws share the common purpose of promoting innovation and enhancing consumer welfare".

[331] Defining predatory innovation and technological tying is a challenging undertaking (see MARIA LILLÀ MONTAGNANI, *Predatory and Exclusionary Innovation: Which Legal Standard for Software Integration in the Context of Competition v. Intellectual Property Rights Clash?*, 37 International Review of Intellectual Property and Competition Law, 304 (2006)), but we should compare this solution with the major clashes between IP and competition policy which could arise from a disclosure obligation.

[332] As some commentators would likely describe the European Decision concerning Microsoft case.

[333] The "essential facility" would be the set of pieces of information needed to achieve interoperability with a dominant platform (and not the entire source code of the platform). Part of the literature focuses its attention on the related problem of "unilateral refusal to license" IP rights.

Bibliography

JOHN ABBOT, *Reverse Engineering of Software: Copyright and Interoperability*, 14 J.L. & Inf. Sci., 7 (2003)
MARK ARMSTRONG, *Competition in Two-Sided Markets*, 37 RAND Journal of Economics, 668--691 (2006)
JAMES BESSEN & ROBERT M. HUNT, *An Empirical Look at Software Patents*, 16 Journal of Economics and Management Strategy, 157--189 (2007)
JAMES BESSEN & MICHAEL J. MEURER, *Patent Failure*, (James Bessen ed., Princeton University Press. 2008)
BRETT A. CARLSON, *On the Wrong Track: A Response to the Manifesto and a Critique of Sui Generis Software Protection*, 37 Jurimetrics J., 187 (1997)
MICHAEL CHAPIN, *Sharing the Interoperability Ball on the Software Patent Playground*, 14 Boston University Journal of Science and Technology Law, 220 (2008)
W. R. CORNISH, *Inter-operable Systems and Copyright*, 11 European Intellectual Property Review, 391--393 (1989)
THOMAS F. COTTER, *Fair Use And Copyright Overenforcement*, 93 Iowa Law Review, 1271 (2008)
K. W. DAM, *Some Economic Considerations in the Intellectual Property Protection of Software*, 24 The Journal of Legal Studies, 321--377 (1995)
BEN DEPOORTER & FRANCESCO PARISI, *Fair use and copyright protection: a price theory explanation*, 21 International Review of Law and Economics, 453—473 (2002)
ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 1*, 22 European Intellectual Property Review, 7-16 (2000)
ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 2*, 22 European Intellectual Property Review, 56-68 (2000)
LOTHAR DETERMANN, *Dangerous Liaisons -- Software Combinations As Derivative Works? Distribution, Installation, And Execution Of Linked Programs Under Copyright Law, Commercial Licenses, And The Gpl*, 21 Berkeley Technology Law Journal, 1421 (2006)
ROBERTO DI COSMO, *Legal Tools to Protect Software: Choosing the Right One*, 4 UPGRADE (2003)
JOSEF DREXL, *IMS Health and Trinko - Antitrust Placebo for Consumers Instead of Sound Economics in Refusal-to-Deal Cases*, 35 International Review of Intellectual Property and Competition Law, 788--808 (2004)
DAVID S. EVANS, et al., *Invisible Engines -- How Software Platforms Drive Innovation and Transform Industries*, (David S. Evans ed., MIT Press First paperback ed. 2008)
JOSEPH FARRELL & PHILIP J. WEISER, *Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age*, 17 Harvard Journal of Law & Technology, 85 (2003)
BRIAN FITZGERALD, *Intellectual Property Rights in Digital Architecture (Including Software): The Question of Digital Diversity*, 23 European Intellectual Property Review, 121--127 (2001)
YVES GAUBIAC, *Interopérabilité et Droit de Propriété Intellectuelle (with en. translation: Interoperability in Intellectual Property Law)*, 211 Revu Internationale du Droit d'Auteur, 91--139 (2007)
GUSTAVO GHIDINI, *Profili evolutivi del diritto industriale. Proprietà intellettuale e concorrenza*, (Giuffrè, Milano. 2001)
GUSTAVO GHIDINI, *Intellectual Property and Competition Law. The Innovation Nexus*, (Edward Elgar. 2006)
SHUBHA GHOSH, *Legal Code and the Need for a Broader Functionality Doctrine in Copyright*, 50 Journal of the Copyright Society of the U.S.A., 71 (2003)
JANE C. GINSBURG, *Four Reasons and a Paradox: The Manifest Superiority of Copyright over Sui Generis Protection of Computer Software*, 94 Columbia Law Review, 2559--2572 (1994)
PAUL GOLDSTEIN, *Comments on a Manifesto Concerning the Legal Protection of Computer Programs*, 94 Columbia Law Review, 2573 (1994)
WENDY J. GORDON, *Fair use as market failure: A structural and economic analysis of the Betamax case and its predecessors*, 82 Columbia Law Review, 1600 (1982)
GIOVANNI GUGLIELMETTI, *Analisi e decompilazione dei programmi*, *in* La legge sul software, 152--201 (Luigi Carlo Ubertazzi ed., 1994)
GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè first ed, Milano. 1996)
GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè second ed, Milano. 1997)
R. J. HART, *Interoperability Information and the Microsoft Decision*, 28 European Intellectual Property Review, 361--365 (2006)
A. JOHNSON-LAIRD, *Software Reverse Engineering in the Real World*, 19 University of Dayton Law Review, 843 (1994)

K. JUDGE, *Rethinking Copyright Misuse*, 57 Stanford Law Review, 901--952 (2004)
DENNIS S. KARJALA, *Copyright Protection of Computer Documents, Reverse Engineering, and Professor Miller*, 19 University of Dayton Law Review, 975 (1994)
ZENTARO KITAGAWA, *Comments on 'A Manifesto concerning the Legal Protection of Computer Programs'*, 94 Columbia Law Review, 2610--2620 (1994)
MARK A. LEMLEY & DAVID W. O'BRIEN, *Encouraging Software Reuse*, 49 Stanford Law Review, 255--304 (1997)
ANNE LEPAGE, *Overview of Exceptions and Limitations to Copyright in the Digital Environment*, January - March UNESCO e-Copyright Bulletin, 1--19 (2003)
JOSH LERNER & JEAN TIROLE, *Some Simple Economics of Open Source*, 50 The Journal of Industrial Economics, 197--234 (2002)
JACQUELINE LIPTON, *The Law of Unintended Consequences: The Digital Millennium Copyright Act and Interoperability*, 62 Washington and Lee Law Review, 487 (2005)
R. MASHIMA, *Examination of the Interrelationship among Japanese I.P. Protection for Software, the Software Industry, and Keiretsu, Part I*, 82 J. Pat. & Trademark Off. Society, 33 (2000)
STEPHEN M. MAURER & SUZANNE SCOTCHMER, *Open Source Software: The New Intellectual Property Paradigm*, NBER Working Paper No. 12148 (March, 2006)
C. R. MCMANIS, *Intellectual Property Protection and Reverse Engineering of Computer Programs in the United States and the European Community*, 8 Berkeley Technology Law Journal, 25 (1993)
C. R. MCMANIS, *Taking Trips on the Information Superhighway: International Intellectual Property Protection and Emerging Computer Technology*, 41 Villanova Law Review, 207 (1996)
CHARLES R. MCMANIS, *The Privatization (or "Shrink-Wrapping") of American Copyright Law*, 87 California Law Review, 173--190 (1999)
PETER S. MENELL, *The Challenges of Reforming Intellectual Property Protection for Computer Software*, 94 Columbia Law Review, 2644 (1994)
A. R. MILLER, *Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU?*, 106 Harvard Law Review, 977 (1993)
MARIA LILLÀ MONTAGNANI, *Predatory and Exclusionary Innovation: Which Legal Standard for Software Integration in the Context of Competition v. Intellectual Property Rights Clash?*, 37 International Review of Intellectual Property and Competition Law, 304 (2006)
KENICHI NAKANO & OSAMU HIRAKAWA, *Copyright Protection of Computer 'Interfaces' in Japan*, 12 European Intellectual Property Review, 46--57 (1990)
RICHARD R. NELSON, *Intellectual Property Protection for Cumulative Systems Technology*, 94 Columbia Law Review, 2674 (1994)
N. T. NIKOLINAKOS, *The New Legal Framework for Digital Gateways – the Complementary Nature of Competition Law and Sector-specific Regulation*, 9 European Competition Law Review, 408--414 (2000)
ANDREA OTTOLIA & DAN WIELSCH, *Mapping the Information Environment: Legal Aspects of Modularization and Digitalization*, 6 Yale Journal of Law and Technology, 174 (2004)
EFTHIMIOS PARASIDIS, *A Sum Greater than Its Parts? Copyright Protection for Application Program Interfaces*, 14 Texas Intellectual Property Law Journal, 59 (2005)
T. A. PIRAINO, JR., *Identifying Monopolists' Illegal Conduct Under the Sherman Act*, 75 New York University Law Review, 809 (2000)
J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Columbia Law Review, 2432 (1994)
J. C. ROCHET & J. TIROLE, *Platform Competition in Two-Sided Markets*, 1 Journal of the European Economic Association, 990--1029 (2003)
J. C. ROCHET & J. TIROLE, *Two-Sided Markets: An Overview*, IDEI Toulouse working paper (March, 2004)
J. C. ROCHET & J. TIROLE, *Two-Sided Markets: A Progress Report*, 37 RAND Journal of Economics, 645--667 (2006)
BORIS ROTENBERG, *The Legal Regulation of Software Interoperability in the EU*, NYU School of Law, Jean Monnet Working Paper 07/05 (2005)
EDWARD SAMUELS, *The Idea-Expression Dichotomy in Copyright Law*, 56 Tennessee Law Review, 321 (1989)
PAMULE SAMUELSON & S. SCOTCHMER, *The Law and Economics of Reverse Engineering*, 111 Yale Law Journal, 1575--1663 (2002)
PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, 94 Columbia Law Review, 2308--2431 (1994)

NOAM SHEMTOV, *Rethinking Entitlements in the Context of Decompilation of Computer Programs: A Market-Based Perspective*, Working paper presented at the 3 rd Annual Workshop on the Law and Economics of Intellectual Property and Information Technology, 5-6 July 2007, Queen Mary, University of London (July, 2007)
JAAP H. SPOOR, *Copyright Protection and Reverse Engineering of Software: Implementation and Effects of the EC Directive*, 19 U. Dayton L. Rev., 1063 (1994)
RICHARD H. STERN, *Reverse Engineering for Future Compatibility*, 1 European Intellectual Property Review, 175--180 (1994)
K. SUGIYAMA, *Reverse Engineering and Other Issues of Software Protection in Japan*, 11 European Intellectual Property Review, 395 (1991)
JEAN TIROLE, *The Theory of Industrial Organization* (MIT Press, Cambridge: Mass. 1988)
J. E. TITUS, *Right to Reverse Engineer Software: Is Japan Next and Does It Really Matter?*, 19 North Carolina Journal of International Law and Commercial Regulation, 491 (1994)
MIKKO VÄLIMÄKI, *Software Interoperability and Intellectual Property Policy in Europe*, 3 European Review of Political Technologies, 1--11 (2005)
MIKKO VÄLIMÄKI & VILLE OKSANEN, *Patents on Compatibility Standards and Open Source – Do Patent Law Exceptions and Royalty-Free Requirements Make Sense?*, 2 SCRIPT-ed (2005)
P. J. WEISER, *The Internet, Innovation, and Intellectual Property Policy*, 103 Columbia Law Review, 534--613 (2003)

SOFTWARE REVERSE ENGINEERING AND OPEN SOURCE SOFTWARE
Do we need more FUD to be satiated?

Second paper of the dissertation project:
Software Interoperability: Issues at the Intersection between Intellectual Property and Competition Policy

Federico Morando
(federico.morando@email.it)

Ph.D. Programme in Comparative Analysis of Law, Economics and Institutions

October 12, 2009

The Interuniversity Centre for the Comparative Analysis of Law and Economics, Economics of Law, Economics of Institutions

SOFTWARE REVERSE ENGINEERING AND OPEN SOURCE SOFTWARE
Do we need more FUD to be satiated?

ABSTRACT

This paper analyzes legal and economic issues related to the technical possibility of accessing interoperability information through reverse engineering (and software decompilation in particular). In this paper, I offer a critique of legal restraints on software reverse engineering in Europe and of similar restraints in the US, in particular in the context of the Digital Millennium Copyright Act.

Through an analysis of entry conditions for late comers and of the comparative costs of developing programs in the first place or reverse engineering them, the paper shows that limitations on decompilation imposed by article 6 of the Software Directive were mostly superfluous and basically non-binding at the time of drafting. What is more, the paper shows that nowadays, new – and largely unanticipated – developments in the models of software development make these restraints an obstacle to competition against dominant incumbents controlling software platforms. In fact, limitations on the freedom to decompile obstruct major reverse engineering projects performed in a decentralized way, as in the context of an open source community. Hence, since open source projects are the most credible tools to recreate some competitive pressure in several crucial software markets, the paper recommends creating a simpler and clear-cut safe harbor for software reverse engineering.

Existing limits to software decompilation are not likely to cause major failures in software applications markets, but they risk increasing the already high stickiness of relative positions in these markets, *de facto* protecting market leaders against potentially dangerous (frequently open source) competitors, which could bring significant innovation and dynamism into the market. Hence, this paper argues for a simplified and clear-cut safe harbor for software reverse engineering. Such a safe harbor should apply, at least, to any decompilation project aiming at the achievement of interoperability, but legal certainty would benefit from a more general, clear-cut rule and no major, negative effects on innovation may be expected. The claim that a simpler safe harbor for decompilation would not create market failures is supported through the aforementioned analysis of entry conditions in software markets, complemented by references to the literature on decompilation. This literature demonstrates that the cost of software reverse engineering is likely to be very high (even as a proportion of the development cost of the reverse engineered product). The discussion of some case studies, proposed in the first part of the paper, supports these findings.

Of course, the paper confirms that some limitations to the use of the products of reverse engineering are appropriate, but these limitations can derive from a quite traditional analysis of copyright infringement and substantial similarity in expression (economically speaking, verifying that there is no – or very limited – free riding on the sunk development costs of the original creator). In other words, an application of copyright, following the recommendation of the first paper of this dissertation, would offer a sufficient stimulus for innovation also in presence of a clear-cut safe harbor for software reverse engineering.

PAPER 2 - TABLE OF CONTENTS

1. Introduction

Reverse engineering, software patents, the legal status of interfaces: in all these fields a certain degree of fear, uncertainty and doubt (FUD)[1] is spread across the market (and sometimes among scholars, both in the field of intellectual property and competition policy[2]). Given the economic relevance of the software industry (both in itself and as a tool for other productive sectors) this is already quite worrying, but what should worry us more is that the expression "FUD" was already used in this context about twenty years ago[3], and there is no foreseeable end to its appropriateness in the field of legal protection of software, especially for issues related to interoperability.

If we survived twenty years with FUD, one could argue, this problem must be a quite manageable one; after all, we have been able to coexist with it, while the software industry, in its various components, was flourishing. That, in fact, is true: the software industry will not stop growing and producing impressive technological advances just because of some legal uncertainty. However, one should also consider that – during the last two decades – the leadership in this industry has been increasingly taken by a few players and one in particular in the field of operating systems (namely: Microsoft), which has been facing – with increasing frequency – antitrust challenges all over the world (apparently without worrying too much about them).[4] All that may be the effect of historical accidents; however, I suggest that it may also dependent on the fact that legal systems are hindering late comers, trying to use self-help in order to surpass the formidable barriers to entry that protect established software platforms. In other words, in the paper at hand, I will try to show that legal systems and intellectual property law in particular, do not completely do their job in making self-help a viable option for potential entrant in any highly concentrated software market.[5]

My analysis will start showing how relevant reverse engineering is in posing a relevant competitive threat to established software platforms. In order to do that, I will start from introducing – within the Section at hand – the open source model of software development, providing some anecdotal evidence concerning its empirical relevance. Section 2 and its subsections will be devoted to the description of some real-world "case studies". In section 3, I will describe some basic economic principles underlying the reverse engineering activity. This section will also show how costly and imperfect software reverse engineering still is. Section 4 contains the main original contribution of the paper at hand. There, I will specifically address the legal solutions, with respect to the issue of software reverse engineering, proposed by the European Union. The section will analyze the letter of the Software Directive, and article 6 in particular, sketch its legislative history and – what is more relevant – highlight a major limit of this exception, which has been neglected so far by the literature. In fact, I will argue that the so-called "decompilation exception" is especially unfit to be enjoyed by open source communities of developers, which likely represent the most credible competitive threats to the sclerotization of dominant positions in software platforms markets. As section 5 will show, some of the limits of article 6 – even though to a lesser extent – may be found in the approach that some US courts took with respect to software reverse engineering. Moreover, article 6 of the Software Directive directly inspired the interoperability exception of the DMCA, which – as a consequence – shares several of the same pitfalls. Section 6 will draw some consequences from the more descriptive part of the paper and highlight some potential problems arising from the need to coordinate rules related to the reverse engineering of software

---

[1] See LOTHAR DETERMANN, *Dangerous Liaisons -- Software Combinations As Derivative Works? Distribution, Installation, And Execution Of Linked Programs Under Copyright Law, Commercial Licenses, And The Gpl*, 21 Berkeley Technology Law Journal, 1421 (2006), fn 3. For a more general (but non-academic) discussion of the concept, see http://en.wikipedia.org/wiki/Fear%2C_uncertainty_and_doubt.

[2] As an example of a rightly renomed scholar making some confusion (or, if you prefer, spreading the confusion created by Microsoft), see FRANÇOIS LÉVÊQUE, *Innovation, Leveraging and Essential Facilities: Interoperability Licensing in the EU Microsoft Case*, 28 World Competition, 71--91 (2005), p. 71, writing about the European Microsoft case that "[a]ccording to Microsoft [a certain interface information] is protected by several patents and such a protection justifies its refusal to disclose." Now, if something was patent protected, disclosing it would likely be a requirement for patentability and, in any case, such a disclosure would not imply any right of third parties to free ride on the disclosed information. In this case, it would be much more sensible to argue that precisely because this information is not patent protected, Microsoft does not want to disclose it, in order to avoid the possible free riding of competitors on its trade secrets.

[3] At least, it was used about 19 years ago in the debate preceding the adoption of the European Software Directive. *See* W. R. CORNISH, *Inter-operable Systems and Copyright*, 11 European Intellectual Property Review, 391--393 (1989).

[4] I am making reference, in particular, to Microsoft, that faced antitrust prosecution in several jurisdictions (in the US, in the European Union, in Japan, in Korea, etc.).

[5] In the third and last paper of the dissertation at hand, I will also address some crucial competition policy issues, which are especially relevant for software market. In fact, FUD may be intentionally created by companies as a competitive move, which could be subject to antitrust scrutiny.

with both copyright and patent law. The fact that, for the time being, this coordination seems to be still quite manageable should not inspire an excessive minimization of the problem. In fact, for the moment, patent law still plays a relatively minor role in protecting software innovation.[6] However, already some years ago,[7] the cross-veto power coming from software patents started representing (and, in perspective, will surely represent even more) a deadly threat for some models of software development and may create a true "tragedy of the anti-commons".[8] Once again, this problem may be especially severe for open source software developers and for small software developers in general. In fact, problems arising from this cross-veto power may be eased by blanket cross-licensing among big software firms (holding significant patent pools), but this solution is itself creating problems, significantly increasing barriers to entry in the software industry.[9] Finally, section 6 will touch on some legislative developments that more or less explicitly took into account the specificities of the open source movement in drafting (or rejecting) modifications to intellectual property law in the field of intellectual property. Section 7 will highlight the relevance of the open source model of software development for the competitiveness of the software industry. This section will show that the law, far from creating special disadvantages for this model of software development, should support it (or, at least, avoid its discrimination). Finally, section 8 will conclude, recommending a clear-cut safe harbor for software decompilation.

1.1. An anecdotal introduction to the open source model of software development

Tuesday the 17th of June 2008 has been a symbolically quite significant day for the worldwide open source (or FLOSS)[10] community. On that day, the latest version of the most popular open source browser, Mozilla's

---

[6] Both the US and the European Patent Offices are quite generous in granting software-related patents (even though with some significant differences, as I will discuss), however there are reasons to believe that the majority of these patents are, in fact, related to high-tech industries making use of software more than to the software industry itself. About this point, see in particular JAMES BESSEN & MICHAEL J. MEURER, *Patent Failure*, (James Bessen ed., Princeton University Press. 2008), chapter 9 (and § 9.1.1 in particular): "[C]redible evidence shows that software publishers have flourished so far despite the growth of software patents. Some preliminary studies suggest that although there is evidence of some detrimental effects of patents within the software industry, the effect of patents within the industry has not been serious to date. […]. [However, the arguments of authors who maintain that anxieties about software patents are exaggerated] seem to be largely directed at a carefully chosen straw man. The general concern is over software patents, not the software industry per se. This distinction is important because almost all software patents are obtained by firms outside the software industry. Bessen and Hunt (2007) find that the software publishing industry only obtains 5% of all software patents granted; most are obtained by firms in electronics, telecommunications and computer industries. […] Patents have little negative effect within the software publishing industry to date because there are no substantial patent thickets within the industry." See also JAMES BESSEN & ROBERT M. HUNT, *An Empirical Look at Software Patents*, 16 Journal of Economics and Management Strategy, 157--189 (2007) (or JAMES BESSEN & ROBERT M. HUNT, *An Empirical Look at Software Patents*, Federal Reserve Bank of Philadelphia Working Papers 03-17/R (March, 2004)), collecting "evidence that software patents *substitute* for R&D at the firm level; they are associated with *lower* R&D intensity."

[7] Starting between 1998 and 2000 on both sides of the Atlantic, according, for instance, to MARCO RICOLFI, *Is There an Antitrust Antidote Against IP Overprotection within Trips?*, 10 Marq. Intell. Prop. L. Rev., 305--367 (2006), pp. 355—363 in particular. The author mentions the following cases as the starting point of the "parallel development" of patent protection of software in the US and in the EU: State Street Bank & Trust v. Signature Fin. Servs., 149 F.3d 1368 (Fed. Cir. 1998), cert. denied, 119 S. Ct. 851 (1999); and In re Int'l Bus. Mach. Corp., 31 IIC 189 (2000).

[8] I will not discuss the literature about the anti-commons in this paper. For further references, see in particular: MICHAEL A. HELLER, *The Tragedy of the Anticommons: Property in the Transition from Marx to Markets*, 111 Harvard Law Review, 621--687 (1998); MICHAEL A. HELLER & REBECCA S. EISENBERG, *Can Patents Deter Innovation? The Anticommons in Biomedical Research*, 280 Science, 698--701 (1998); J. BUCHANAN & Y. YOON, *Symmetric Tragedies: Commons and Anticommons Property*, 43 Journal of Law and Economics, 1-13 (2000). For further references, see the third paper of the dissertation at hand and the following works: BEN DEPOORTER & FRANCESCO PARISI, *The Market for Intellectual Property: The Case of Complementary Oligopoly*, *in* The Economics of Copyright: Developments in Research and Analysis, (W. Gordon & R. Watt eds., 2003); FRANCESCO PARISI, et al., *Simultaneous and Sequential Anticommons*, 17 European Journal of Law and Economics, 175--190 (2004); GIUSEPPE DARI-MATTIACCI & FRANCESCO PARISI, *Substituting Complements*, 2 Journal of Competition Law and Economics, 333--347 (2006).

[9] See the first paper of the dissertation, § 9.2.1 for some examples of cross-licensing to solve the cross-veto power coming from pools of software patents. Obviously, this kind of agreements, are not only apt to increase barriers to entry; they are also unable to eliminate costs arising from patent trolls, essentially patenting software solutions, without really applying them to the development of their own programs.

[10] The acronym FLOSS stands for Free/Libre Open Source Software and wants to highlight that open source software is not only (and not necessarily) free in monetary terms, but it is "free" as in "freedom". In fact, the Frech work "libre" eliminates the ambiguity of the English word free, meaning also *gratis* (i.e. "for free"). See also http://en.wikipedia.org/wiki/FLOSS. I will frequently use the term FLOSS, because – as described by the Wikipedia – it is "an inclusive term generally synonymous with both free software and open source software which describe similar development models, but with differing cultures and philosophies. 'Free software' focuses on the philosophical freedoms it gives to users and 'open source' focuses on the perceived strengths of its peer-to-peer development model."

Firefox 3.0, was downloaded by 8,002,530 people (or, at least, by as many IP addresses) and Guinness confirmed to Mozilla that Firefox effectively set a new world record as the "most downloaded software in a single day"[11]. This news was reported by several major media, frequently along with a much more significant datum concerning the market share of Firefox in the browser market, according to some estimates having passed 19% worldwide during June 2008[12] (and being around 30% in Europe).

Firefox 3.0 is an interesting example of how powerful an open source community can be in competing with established incumbents. In January 1998, at the end of the so-called "browser war", preceding and accompanying the famous US Microsoft (III) antitrust case, Netscape (before being bought by AOL) started an open source project, called Mozilla[13] (using a peculiar license[14] and keeping to itself the possibility of releasing proprietary versions of Netscape). About 10 years later, on March 1st, 2008, support for Netscape browser from AOL officially ended. But the "browser war" had been won by Microsoft several years earlier, and that victory was sanctioned (and, in a way, "paid") by Microsoft with the settlement it signed, on May 2003, with AOL, which had previously filed suit for damages (on the basis of the finding of facts in the Microsoft III antitrust case).[15] About at the same time, the Mozilla project announced that it was going to focus its attention on two projects, the main one being Mozilla Firefox (and the other Thunderbird, an open source mail client). In fact, not only the end of the browser war between Netscape and Microsoft's Internet Explorer (IE) coincides – in some way – with the beginning of a "second browser war" between IE and Firefox, but the very source code of Netscape, passing through the Mozilla suite, is in some way an ancestor of Firefox's code (even thought, in the meantime, it has likely changed enough as to maintain very pale similarities).[16]

For the purposes of the paper at hand, the point of this story is that – in order to compete with big software houses, in markets that they consider as strategic and in which they are willing to invest a lot of resources (and maybe risk antitrust liability) – it is frequently useful, (if not necessary), to ask the help of the open source community. Indeed, a very similar story could be told about Sun Microsystems' Star Office, the source code of which is behind the success of the famous OpenOffice suite, competing with Microsoft Office.

I will elaborate more on this point later (in § 7). For the moment, what I want to argue is just that open source projects are sometimes better able to compete with proprietary ones, thanks to their different structure of incentives (and lack of traditional financial budget constraints). Moreover, I submit that the success of Firefox and other open source software, frequently running (also) on Windows, is lowering the switching costs potentially associated with leaving the dominant Windows operating system. And that is true because these pieces of software are available for the majority of other platforms (Linux, Mac, etc.), hence they make it much easier to migrate one's data and settings (or even to share them running two or more platforms on the same computer, as on the PC on which this paper was written).

Discussing these switching costs, the same day as Firefox's 3 launch something else happened that is likely to influence even more the level of the barriers to entry protecting Window's sales. Something that is especially relevant for the paper at hand. In fact, on June 17th, 2008, the team of Wine's developers released version 1.0 of Wine. Building this piece of software to a stable version took 15 years of development. But what is Wine, why is it relevant, and why will we probably heard more about it? Wine allows Linux and other Unix-like operating systems users to install and run applications developed for various versions of Microsoft Windows (notice that this implies that also users of recent Mac systems can use Wine). As an illustration, I wrote part of this paper on a copy of Microsoft Word installed on a Linux Kubuntu system, needing no more

---

[11] See Maggie Shiels, *Firefox download record official*, Technology reporter, BBC News, Silicon Valley, 2008/07/03 (available on http://news.bbc.co.uk/go/pr/fr/-/1/hi/technology/7486668.stm; last visited July 13, 2008).

[12] Statistics provided by Net Applications (http://marketshare.hitslink.com/) for the month of June 2008. The trend reported by Net Applications is constantly increasing for Firefox in the last 2 years. In July, 2006 Internet Explorer had 83.57% of the global market (considering all operating systems), Firefox 11.34% and (Apple's) Safari 3.18%. In June 2008, Internet Explorer is reported at 73.01%, Firefox at 19.03%, Safari at 6.31%.

[13] For a short history of Netscape, see http://en.wikipedia.org/wiki/Netscape (last visited July 27, 2008).

[14] The "Netscape Public License" (NPL), available at http://www.mozilla.org/MPL/NPL-1.0.html.

[15] With *Microsoft III* I refer to the antitrust case, which risked leading to the dismembering of Microsoft, following J. Jackson's ruling 87 F.Supp.2d 30 (D.D.C., 2000). In appeal, the case was vacated and remanded, with ruling 253 F.3d 34 (C.A.D.C., 2001). The case has been ended by the *Consent Decree* ratified by J. Kollar-Kotelly, with ruling 231 F.Supp.2d 144 (D.D.C., 2002). Consider also that with *J. Jackson's findings of fact* I refer to ruling 84 F.Supp.2d (D.D.C., 1999).

[16] It is probably not by chance that Firefox's original name (changed because of trademark issues) was Phoenix, hinting to the fact that it had been reborn from the Navigator's ashes. See http://en.wikipedia.org/wiki/Mozilla_Firefox#History for more details.

than half an hour to complete the installation of both the latest available version of Wine[17] and an original copy of Microsoft Office 2003. Similarly, no customization or special effort was needed to install a piece of econometric software that economists frequently use: Stata 9 for Windows.[18]

The relevance of Wine and similar projects (that I will discuss in greater detail later on) is that they could significantly lower the so-called "application barrier to entry" that protects the leading PC operating system, i.e. Microsoft's Windows. In fact, having the possibility of installing copies of one's favorite (or must-have, for any reason) software on Linux (or other Unix "dialects", FreeBSD, Mac OS X and Solaris) evidently decrease the total switching cost of changing operating system. That is true for various reasons: not only does one not have to buy a new license for a Linux version (if existing) of a software that one already owns; but also perfect compatibility of saved data is assured (and that would not always be the case using open source alternative – or even different versions of the same commercial package – for some kinds of data, including – for instance – word processor files with very complex formatting).

To keep things in perspective, notice that, for Microsoft and similar incumbents, the threat coming from similar projects is significant, but not necessarily deadly, in particular if they keep innovating at a fast pace. In fact, one could reasonably suspect that the fact that Wine was recently able to reach a stable version is also related to the slowdown in the pace of innovation for computer operating systems. In particular, the penultimate version of Windows, Windows XP, has probably shown the highest longevity in the history of Microsoft's operating systems, having being released in October 2001 and still being the most widespread operating system, with a market share above 70% as of June 2008[19] (in fact, Microsoft recently committed to support Windows XP until 2014[20]). Moreover, it must be noted that the compatibility between Unix-like systems and applications developed for Windows is still far from perfect. In fact, perfect compatibility (and stability of applications used outside their "natural environment") is a very complex goal, frequently incompletely reached even by new versions of the same operating system (several computer users, for instance, are aware of more than one problem in running on Windows Vista applications developed for Windows XP[21]). And, to be sure, for the moment there are many more problems in trying to use on Linux an application developed mainly for Windows XP than there are problems in doing so on Windows Vista, hence the application barrier to entry did not disappear, and will probably not disappear in the following decades; it has just been significantly lowered, so that less and less users will find it prohibitive, as long as they have other reasons to shift from one operating system to another.

What this anecdotal introduction aimed at showing is simply that open source software is, by now, a credible competitor for incumbent software houses in several markets. Moreover, it is important to be aware of the fact that an open source approach is adopted with increasing frequency by commercial software houses, precisely in those cases in which they decide to fight (with apparently dim chances of success) against established incumbents. Unfortunately, the fact that open source development may be one of the most effective tools to inject competition into especially concentrated software markets should worry us, since – as I will show – some peculiarities of the legislation on software reverse engineering could put this specific model of software development at a disadvantage in achieving interoperability through software decompilation.

---

[17] At the time of writing these lines, the latest available version of Wine was Wine 1.1.1, released July 11, 2008. The official website (http://www.winehq.org/) claims that this version fixed bugs preventing the working of Photoshop CS3 and Office 2007 installers (I could not verify this claim or the working of the installed software, since I do not own a copy of these recent "best seller" programs).

[18] On the Wine Applications database (http://appdb.winehq.org/), Stata 9 for Windows is reported to be randomly unstable under older versions of Wine. I did not test this thoroughly, but this is largely irrelevant, since there is an official version of Stata for Linux.

[19] According to Net Applications' "Operating System Market Share", Windows XP has a 71.20% market share, as of June 2008, while Windows Vista at 16.14%. The total share of Windows operating systems is 90.89% (See http://marketshare.hitslink.com/report.aspx?qprid=10 for updated statistics. Last visited July 27, 2008).

[20] See http://support.microsoft.com/lifecycle/?LN=en-gb&C2=1173 (last visited July, 27, 2008). See also Paul McDougall, *Microsoft Pledges Windows XP Support Through 2014*, InformationWeek, June 24, 2008 (available at http://www.informationweek.com/news/windows/operatingsystems/showArticle.jhtml?articleID=208800494. Last visited, July 27, 2008).

[21] For a brief summary of the various opinion concerning Windows Vista and its compatibility problems, see – for instance – Paul McDougall, Microsoft Calls Forrester's Windows Vista Report 'Schizophrenic', InformationWeek, July 28, 2008 (available at http://www.informationweek.com/news/windows/operatingsystems/showArticle.jhtml?articleID=209602050 ; last visited July 28, 2008).

1.2. Summary of relevant points (and definitions) discussed in the first paper

This paper assumes that the reader has a good familiarity with the issues discussed in the first paper of the dissertation project. Here I will just recall a few concepts, for the sake of completeness, but readers familiar with the first paper of this dissertation may safely go quickly through the present section (or possibly jump directly to § 1.3.1).

About software reverse engineering in general, readers should be aware of the practical impossibility of reading and understanding the object (or compiled/binary) code of software programs. The humanly realized source code of computer programs is transformed in the object code, which can directly run on a computer, by the operation called compilation. Evidently, decompilation[22] is the inverse operation, which – however – is a complex process and cannot reconstruct several elements (comments, name of variables, etc.) which are precious for the understanding of a program, but superfluous for its execution by the machine. Moreover, the decompilation procedure cannot be completely performed in an automatic way, differently from the compilation one. I will come back to this topic later. For the moment, in order to favor an intuitive understanding of the topic, I just propose a suggestive metaphor of software reverse engineering offered by Band & Katoh:

> In sum, reverse engineering source code is like reverse engineering a Lexus off the Showroom floor; the programmer, lust like the General Motors engineer, can examine the program methodically, figuring out how the program works and how it interfaces with other elements of the computer. In contrast, reverse engineering object code is like putting the General Motors engineer in a garage in which all the parts of a completely disassembled Lexus had been scattered haphazardly. Before he can begin to figure out how the Lexus works, he must substantially reassemble most of its parts so that he can see what they look like. Fortunately for General Motors, an unassembled Lexus cannot operate, so Lexus sells its cars fully assembled. Conversely, computers can execute object code, so vendors distribute programs in incomprehensible object code format.[23]

Not only is software reverse engineering complex. It is also possible to use some techniques to make decompilation even more difficult than it normally is. That is achieved, for instance, using software tools to produce "obfuscated code" (or "shrouded code")[24]. This kind of code, though still producing the same results as the original code (when it runs on a computer), is intentionally structured in such a way that it is very difficult to understand for human beings. If these techniques are used, automatic decompilation leads to results, which are especially far from being immediately usable, as a guideline to develop a functionally equivalent program. Realizing a non-copyright-infringing functional equivalent of a decompiled program is always a complex task, needing the intervention of skilled programmers; however, when obfuscation techniques are used, even once the initial machine readable code has been transformed in a human readable code, an authentic additional reverse engineering activity is needed in order to re-structure the source code in an intelligible way.

That having been said, it is also important to recall how an Application Programming Interface (API) works[25]. In a few words, APIs function in a similar way as mathematical functions in a spreadsheet. For instance, users do not need to create a specific procedure in order to get a number rounded to a specific number of digits; instead, they just type the function respecting a given syntax and the program returns the result. E.g. "round(*number*, *digits*)", where *number* is obviously the number to round (or the reference to its cell) and digits is an integer specifying how many digits the software should keep to the right of the dot. Obviously, the actual algorithm to round the number may vary from one spreadsheet to another and also the syntax may be slightly different. Quite clearly, at least in this simple case, the "form" of the interface – function(*argument1*, *argument2*) – is not creative at all. However, in order to take advantage of the existing function the user needs to know that it exists and which syntax should be used to ask the software to perform

---

[22] The first passage from object (binary) machine code to the low level instructions for the computer processor is called "disassembly" (because this is the inverse of the work of an assembler, transforming instruction in the assembly low level language into executable code). I add this note just because the term disassembly is sometimes (improperly) used as a synonym of the entire decompilation effort. For more technical information, see the Wikipedia's pages on this topic and, in particular, about decompilation: http://en.wikipedia.org/wiki/Disassembly; http://en.wikipedia.org/wiki/Decompilation.

[23] JONATHAN BAND & MASANOBU KATOH, *Interfaces on Trial -- Intellectual Property and Interoperability in the Global Software Industy*, (Jonathan Band ed., Westview Press First ed, Boulder, Colorado. 1995), p. 14.

[24] See http://en.wikipedia.org/wiki/Obfuscated_code, including various example of obfuscated source code.

[25] For more references, see the first paper of this dissertation and, among many others, DAVID S. EVANS, et al., *Invisible Engines -- How Software Platforms Drive Innovation and Transform Industries*, (David S. Evans ed., MIT Press First paperback ed. 2008), p.27.

this task. In this example, we could say that the "specification" of this function consists in its "external part" (the fact that the software acts in response to a function written as round(*number*, *digits*)) and in a description of the mathematical properties of this operator (e.g. if the first dropped digit is 5 or more, then increase the last shown digit by one; otherwise leave it as it is). The "black box" actually performing the requested operations may be said to be an "implementation" of the previous specification. In fact, there may be several more or less efficient algorithms to actually round numbers on different kinds of computer platforms and these kinds of details pertain to the implementation.

Hence, to take advantage of an API – or of a similar interface – one has to know that it exists and to derive (or guess) its correct syntax. And that is about all that is needed to achieve "vertical interoperability", i.e. to use one's piece of software as a complement of someone else's program. Evidently, to offer a "compatible" interface one has to know something more. In fact, the inner working of the API must be guessed, so that one is sure to be able to give exactly the same response as the original software when it is asked the same service. In some cases, that is quite obvious and can be done just writing down a software implementing some standard mathematical algorithms. In other case, to really achieve perfect interoperability, one has to deeply analyze the original software. In fact, when an API does not work as expected, some reverse engineering may be useful even for vertical interoperability and when the API specification is, in principle, perfectly known. For instance, a given bug in an existing API could call for a modification in the application wanting to be interoperable, if the software "exposing the API" (i.e. offering the interface to other applications) cannot be modified by its developer for any reasons.

In the first paper of the dissertation project at hand, I argued that trade secret may be an economically efficient tool of protection to stimulate software innovation. Borrowing the words of Prof. Reichman, in the field of software and software interfaces in particular: "The exclusive rights regimes as a class are inherently more anticompetitive than the trade secret laws whose defective operations they seek, wittingly or unwittingly, to redress, because the latter impose no insuperable barriers to entry."[26] However, I also stressed that copyright is needed to complement trade secret in the software industry, but copyright itself should mainly provide a tool to exclude software piracy (i.e. literal copying) and other activities based on copying expression. Ideas, principles and methods should remain free, and that is particularly true for interface specifications.

In this paper I will show more thoroughly that the positive role of copyright complemented by trade secret can be fully exploited only as long as reverse engineering is essentially free (as I assumed in the first paper, at least when the legitimate goal of reverse engineering is to achieve interoperability). In fact, in the field of software there is already a double layer of protection: trade secret complemented by copyright. Limiting reverse engineering by law and/or offering legal protection to technologies used to prevent reverse engineering (especially in the field of interoperability) could have a negative impact on welfare. If intellectual property law does prohibit reverse engineering or significantly hinders it, its main purpose – i.e. a dynamic and pro-competitive innovative environment – may not be reached.[27]

1.3. Direct (or vertical) and indirect (or horizontal) interoperability

Reverse engineering is used to achieve both direct (also called vertical) and indirect (or horizontal) interoperability. These concepts have already been introduced in the first paper of the dissertation project and are related to the realization of a complementary product (when direct/vertical interoperability is to be achieved) or of a substitute product (when indirect/horizontal interoperability is pursued). In fact, since achieving direct interoperability creates a new complement for an existing piece of software (increasing its value), information needed to achieve it is frequently easier to acquire and it may be completely withheld in only some special cases.[28] However, the disclosure of this information is frequently subject to some kind of non-disclosure agreements and additional conditions. An easily understandable reason for these limitations may simply be that widespread direct interoperability information makes easier to collect indirect

[26] J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Columbia Law Review, 2432 (1994), p. 2530.

[27] I will not discuss that in the paper at hand, but notice that also superimposing a significant layer of patent protection for software (above the basic level of copyright protection) may have very similar effects (i.e. it may hinder the dynamism and competitiveness of software markets).

[28] Some of these cases are discussed by Farrell and Weiser as "exceptions" to their paradigm of internalization of complementary efficiencies (ICE). *See* JOSEPH FARRELL & PHILIP J. WEISER, *Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age*, 17 Harvard Journal of Law & Technology, 85 (2003).

interoperability information as well.[29] In other cases – and here the best example is provided by game consoles, which are specialized integrated hardware/software platforms – information concerning direct interoperability could be conditioned by the payment of royalties and the respecting of precise quality requirements,[30] because this enables specific pricing policies and business models.

Some examples involving the two kinds of interoperability may clarify the concepts and – what is more important – provide insights for the following reasoning.

1.3.1. Direct (vertical) interoperability

Direct interoperability is a crucial issue for any kind of software. In fact, general purpose operating systems (like Windows, Linux or Mac OS) always guarantee a very high level of direct interoperability to developers wanting to run their applications on these platforms. Their APIs and the main communication protocols used by them are, for the vast majority, quite well documented and their use even incentivized (frequently indirectly, through free websites, more o less cheap software development kits, Internet forums and mailing lists).[31] However, alternative models – in which direct interoperability is tightly controlled – exist and concern other software platforms (different from ordinary personal computers). In particular, this is the case for game consoles and – up to a certain extent (and depending on the producer) – for smart-phones.

As far as personal computers are interested, direct interoperability may be an issue in particular for software drivers. So-called "driver wrappers" have been the main category of software created using reverse engineering and devised to solve direct interoperability problems:

> A driver wrapper is software that functions as an adapter between an operating system and a driver, such as a device driver, that was not designed for that operating system. It can enable the operating system to use technologies for which no native implementation exists.[32]

The most relevant examples of this kind of technology are open source projects allowing the use of Microsoft Windows' device drivers under other operating systems. In particular, driver wrappers allowing the use on Linux of wireless cards developed and distributed just for Microsoft Windows.[33]

> Many vendors do not release specifications of the hardware or provide a Linux driver for their wireless network cards. This project [i.e. NDISwrapper] implements Windows kernel API and NDIS (Network Driver Interface Specification) API within Linux kernel. A Windows driver for wireless network card is then linked to this implementation so that the driver runs natively, as though it is in Windows, without binary emulation.[34]

Another (relatively) famous case concerns the Captive NTFS project[35], which has been one of the first tools allowing read/write access to Windows XP and Windows Vista formatted disk partitions (and the associated files) under Linux.[36] Obviously, in all these cases the original drivers to "wrap" are copyright protected and must be (legally) obtained by the final user in some way (but usually this is not a problem, since a user owning a piece of hardware will also typically own a licensed copy of the associated device driver).

---

[29] To be sure, knowing all the specifications to achieve direct interoperability for any piece of software with a given platform would be sufficient to create a competing platform indirectly compatible with the first one.

[30] More details about the special market structure of game consoles are provided by the literature. *See*, in particular, EVANS, et al., *Invisible Engines*, . A qualitatively equivalent, but much more formalized, analysis may be found in ANDREI HAGIU, *Two-sided Platforms: Pricing and Social Efficiency*, Harvard Business School and Research Institute of Economy Trade and Industry working paper, Cambridge, Mass. (2005). *See* also ANDREI HAGIU, *Pricing and Commitment by Two-Sided Platforms*, 37 Rand Journal of Economics, 720--737 (2006).

[31] Of course, that is not always the case for some specific communication protocols, especially the ones used by workgroup servers (this issue will be discussed below, describing the Samba project) or for some APIs mainly used by applications realized by the same operating system's owner and/or by other strategic applications (like internet browsers and other middleware, which could represent a competitive threat to the operating system itself).

[32] See "Driver Wrapper" on the Wikipedia (available at http://en.wikipedia.org/wiki/Driver_wrapper; last visited: July 26, 2008).

[33] NDISwrapper (http://ndiswrapper.sourceforge.net/) is the name of such a project for Linux.

[34] From the NDISwrapper homepage (http://ndiswrapper.sourceforge.net/) (retrieved July 27, 2008).

[35] See http://www.jankratochvil.net/project/captive/. Notice that (as reported on the project's website) this project uses parts of the code of ReactOS project that I will discuss below.

[36] At present, there are several alternative solutions, including a driver embedded in the Linux kernel itself. More references could be found here: http://en.wikipedia.org/wiki/NTFS.

1.3.2. Indirect (horizontal) interoperability

Achieving horizontal interoperability with an existing software product is a burdensome task, especially if the reverse engineered piece of software is a very complex and huge (in terms of number of lines of code) system, such as an operating system or a software platform in general. However, it is precisely in competing with software platforms that the achievement of horizontal interoperability may be necessary, in order to overcome the so-called "application barrier to entry". This concept has been made famous by the US Microsoft (III) antitrust case. In this context, *Judge Jackson's findings of fact*[37] described in the following way the "intractable 'chicken-and-egg' problem" faced by firms wanting to compete with an established leading software platform, for which thousands of applications have been developed, like Microsoft's Windows:

> The overwhelming majority of consumers will only use a PC operating system for which there already exists a large and varied set of high-quality, full-featured applications, and for which it seems relatively certain that new types of applications and new versions of existing applications will continue to be marketed at pace with those written for other operating systems. Unfortunately for firms whose products do not fit that bill, the porting of applications from one operating system to another is a costly process. Consequently, software developers generally write applications first, and often exclusively, for the operating system that is already used by a dominant share of all PC users. Users do not want to invest in an operating system until it is clear that the system will support generations of applications that will meet their needs, and developers do not want to invest in writing or quickly porting applications for an operating system until it is clear that there will be a sizeable and stable market for it. What is more, consumers who already use one Intel-compatible PC operating system are even less likely than first-time buyers to choose a newcomer to the field, for switching to a new system would require these users to scrap the investment they have made in applications, training, and certain hardware.[38] […]
> [T]he chicken-and-egg problem (hereinafter referred to as the "applications barrier to entry") would make it prohibitively expensive for a new Intel-compatible operating system to attract enough developers and consumers to become a viable alternative to a dominant incumbent in less than a few years.[39]

While discussing the "Empirical Evidence of the Applications Barrier to Entry", Judge Jackson also analyzed the experiences of IBM and the case of IBM's operating system, OS/2 Warp.[40] The first part of the development of OS/2 was conducted in joint venture by Microsoft and IBM and IBM did not really need to decompile a huge amount of Microsoft's code in order to achieve horizontal interoperability, simply because Microsoft was helping this attempt. Despite this advantage – that any "normal" competitor would never enjoy – IBM's efforts to realize a completely interoperable system never succeeded or, at least, were never commercially successful (probably also because Microsoft Windows was much cheaper than IBM's system). IBM discontinued support for OS/2 at the end of 2006 and has no plans for further development[41]. The failure of IBM to realize an operating system, which was supposed to be fully interoperable with Microsoft Windows, but also technically superior, are telling. This case even pushed Judge Jackson to argue that "cloning the 32-Bit Windows APIs" would be virtually impossible or, at least, an economically unbearable effort for any undertaking.[42] In fact, as the history of the software industry seems to confirm, there are

---

[37] With *J. Jackson's findings of fact* I refer to ruling 84 F.Supp.2d (D.D.C., 1999): United States District Court for the District of Columbia, United States of America v. Microsoft Corporation, Civil Action no. 98-1232 (TPJ).

[38] *Judge Jackson's findings of fact*, § 30.

[39] Judge Jackson's Finding of Facts (see supra note 37), § 31.

[40] Judge Jackson's Finding of Facts (see supra note 37), § 46: "IBM's inability to gain widespread developer support for its OS/2 Warp operating system illustrates how the massive Windows installed base makes it prohibitively costly for a rival operating system to attract enough developer support to challenge Windows. In late 1994, IBM introduced its Intel-compatible OS/2 Warp operating system and spent tens of millions of dollars in an effort to attract ISVs [i.e. Independent Software Vendors] to develop applications for OS/2 and in an attempt to reverse-engineer, or "clone," part of the Windows API set. Despite these efforts, IBM could obtain neither significant market share nor ISV support for OS/2 Warp. Thus, although at its peak OS/2 ran approximately 2,500 applications and had 10% of the market for Intel-compatible PC operating systems, IBM ultimately determined that the applications barrier prevented effective competition against Windows 95. For that reason, in 1996 IBM stopped trying to convince ISVs to write for OS/2 Warp. IBM now targets the product at a market niche, namely enterprise customers (mainly banks) that are interested in particular types of application that run on OS/2 Warp. The fact that IBM no longer tries to compete with Windows is evidenced by the fact that it prices OS/2 Warp at about two-and-one-half times the price of Windows 98."

[41] See IBM official website (http://www-306.ibm.com/software/os/warp/withdrawal.html).

[42] Judge Jackson's Finding of Facts (see supra note 37), § 52: "Theoretically, the developer of a non-Microsoft, Intel-compatible PC operating system could circumvent the applications barrier to entry by cloning the APIs exposed by the 32-bit versions of Windows (Windows 9x and Windows NT). Applications written for Windows would then also run on the rival system, and consumers could use the rival system confident in that knowledge. Translating this theory into practice is virtually impossible, however. First of all, cloning the thousands of APIs already exposed by Windows would be an enormously expensive undertaking.

reasons for thinking that cloning the entire set of APIs (and communication protocols) used by an operating system is an economically very risky effort, which few players will ever try. I will come back to this issue. Here I just want to mention that there have been petitions from the small, but active, community of OS/2 developers and users to release the source code of the abandoned OS/2 project as open source[43]. However, "for a variety of business, technical, and legal reasons[44]" (arguably including the fact that part of the original OS/2 code has been licensed from Microsoft, possibly under specific non-disclosure agreements) IBM declined to do so. This is interesting, because it is coherent with the possibility that open source development is one of the few credible ways to proceed in order to pursue horizontal interoperability with a major incumbent platform.

Another, less well-known, commercial attempt to implement Microsoft Windows API was tried by Sun Microsystems. The Wabi project concerned an implementation of the Win16 API specification (used by Windows 3.1 and other Microsoft's operating systems) on Sun's Solaris. Quoting from Wabi 2.2 User's Guide:

> The Wabi™ program is a UNIX® application that enables you to run Microsoft Windows applications on several UNIX operating environments that use the X Window System™. Wabi acts as an interface between the Windows world and the UNIX world, translating the language of Microsoft Windows applications to the language of UNIX and the X Window System.[45]

In fact Wabi's work was one of just translation: Wabi always required an installation of Windows on the same computer in order to work.[46] At the same time, the goal of overcoming the application barrier to entry was explicit, starting from the first chapter of of Wabi's manual:

> The Wabi program lets you enjoy the benefits of the security, power, and connectivity of your UNIX operating system, and on the same desktop, take advantage of popular Microsoft Windows applications such as spreadsheets, word processors, databases, graphics packages, and more.[47]

Wabi's development started in the early '90es and was discontinued in December 1997[48], just a few months after a last attempt from the Linux distributor Caldera to offer Wabi (licensed from Sun) to its customers[49].

It may be interesting to notice that the Wabi project had been coupled, by Sun, with an effort to create an ISO standard, describing a non-proprietary specification of the entire Windows API, called Public Windows Initiative (PWI). Despite the fact that Sun – coherently with the thesis described in the first paper of this dissertation – argued that no intellectual property violation was needed in order to create such a standard specification, Microsoft was able to prevent the project from gaining momentum:

> In 1996 Microsoft Corp was able to shoot down another ECMA[50] standard, the Public Windows Inititaive, at [the stage of ISO vote], thus preventing it from becoming an ISO standard. The PWI was a Sun effort to get Windows APIs put into the public domain. […] Microsoft was able to mount a successful campaign against PWI at ISO on [intellectual property grounds].[51]

More daunting is the fact that Microsoft continually adds APIs to Windows through updates and new versions. By the time a rival finished cloning the APIs currently in existence, Windows would have exposed a multitude of new ones. Since the rival would never catch up, it would never be able to assure consumers that its operating system would run all of the applications written for Windows. IBM discovered this to its dismay in the mid-1990s when it failed, despite a massive investment, to clone a sufficiently large part of the 32-bit Windows APIs. In short, attempting to clone the 32-bit Windows APIs is such an expensive, uncertain undertaking that it fails to present a practical option for a would-be competitor to Windows."

[43] See Slashdot news "IBM Won't Open-Source OS/2" (available at http://slashdot.org/article.pl?sid=08/01/22/0258213 last visited: July 27, 2008).

[44] IBM's Yvonne M. Perkins' Letter of January 16, 2008, as reported by OS/2 World.com (available at http://www.os2world.com/content/view/16595/1/. Last visited July 27, 2008).

[45] Wabi 2.2 User's Guide, Chapter 1, "What is Wabi?". The entire manual is available at http://docs.sun.com/app/docs/doc/802-6306 (last visited July 20, 2008).

[46] IBM's OS/2, instead, offered two different options to use software designed for Windows, only the cheapest of which required an installation of Windows on the same computer.

[47] Wabi 2.2 User's Guide, Ch. 1.

[48] See http://en.wikipedia.org/wiki/Wabi_(software).

[49] See Dwight L. Johnson, *Wabi: Caldera's Solution for Windows Applications*, LinuxJournal.com, June 1st, 1997 (available at http://www.linuxjournal.com/article/2076; last visited July 10, 2008)

[50] "Ecma International is an international, private (membership-based) non-profit standards organization for information and communication systems." See the Wikipedia for further details: http://en.wikipedia.org/wiki/Ecma_International.

[51] William Fellows, Sun Uses ECMA as Path to ISO Java Standardization, Computergram International, May 7, 1999 (available at http://findarticles.com/p/articles/mi_m0CGN/is_1999_May_7/ai_54580586; last visited June 23, 2008).

Once again, that SUN efforts failed is coherent with the presence of significant Fear, Uncertainty and Doubt (FUD) surrounding the legal status of software interoperability information.

As the previous examples showed, commercial efforts to generate horizontal interoperability with an existing complex platform are very risky (not to say doomed to failure, according to past experience). As I will show, using the scant available empirical evidence and some more abstract reasoning, this depends on the cost structure of software development and of reverse engineering (in the form of software decompilation) in particular. However, this does not mean that achieving horizontal interoperability, and hence posing a major competitive threat to established software platforms, is impossible. Some open source projects are actually working in this direction – or, at least – in the direction of dramatically lowering the application barrier to entry: Section 2 of this paper is devoted to them.

2. Open source projects pursuing interoperability with commercial software

In this second section of the paper I will describe what has been achieved in the field of interoperability – and in particular in the field of horizontal interoperability – by some open source projects. The goal of these small "case studies" is to provide the reader with an intuitive understanding of what is going on in this field, before trying to provide a more theoretical economic description of it. After these two steps (case studies and economics) the legal background will be described and – in part – criticized.

As I already hinted, today the most significant active projects trying to replicate Windows' APIs are being conducted by open source programmers and are represented by the Wine "interoperability layer", able to run under Linux and other Unix-like systems, and by the fully-fledged operating system ReactOS (which is, however, still in an earlier stage of development). Interestingly, given the natural openness of these projects (meaning the two mentioned above and some other minor FLOSS projects), there are frequent synergies and mutual inspirations between them. In fact, as I will discuss, open source projects raise specific economic and legal issues, precisely because of these synergies or economies of scope. These issues deserve a special attention, and I will analyze them in what follows.[52]

2.1. Projects using black box analysis and similar techniques

Before focusing on the aforementioned Wine and ReactOS projects, it is appropriate to mention that some software reverse engineering techniques cannot be labeled as decompilation. In particular, there are several techniques based on the observation of the behavior of a given system as a black box: here, developers are just looking at how a given system (an entire PC or a piece of software in a controlled environment) reacts to certain inputs and/or interacts with other systems. The "black box" consisting of the original piece of software (usually in compiled binary form) is not "opened", but it is studied "from outside", trying to understand how it communicates with its "environment". Of course, any software developer looks at the working of competing products, in order to understand their functionalities, user interfaces and – in general – strengths and weaknesses. This kind of study is better labelled as "common sense" than as "black box reverse engineering"; however, in some cases this analysis is performed in controlled environments and in a systematic way. For instance, the developers of the Samba open source project extensively practice a black box reverse engineering technique called network analysis.[53]

The Samba project[54] is aimed at allowing Unix-like systems to interoperate with networks encompassing Windows systems (both working as client and as servers). The aforementioned Network analysis techniques are frequently based on capturing data packets in transit on a given network for later analysis, using a packet sniffer (also known as a network analyzer).[55] In this way, it may be possible to understand how a proprietary network protocol operates. The captured packets mainly consist of information generated by the user for the purpose of testing the network protocol, and just the "formatting of the information" (so to speak) is

[52] See § 2.2.1 and 2.2.2. For the moment, I can anticipate that, from the economic point of view, the fact of sharing part of the costs of reverse engineering could make the economic pressure coming from this activity more serious for incumbents (and potentially more likely to represent a disincentive to invest). While, from the legal point of view (and likely considering the previous economic insight) sharing the "information obtained through reverse engineering" could raise doubts of violation of article 6 of the European Software directive.

[53] One of the leading developers of the Italian Samba team [Simo Sorce] confirmed me that the Samba team "do not consider the Network Analysis as a technique that could be labelled 'reverse engineering' [*sic*, *rectius*: 'decompilation'] in the sense of the various legislative texts" [likely thinking in particular to the EU Software Directive, which was the main argument of discussion] (my tentative translation).

[54] See the Wikipedia (http://en.wikipedia.org/wiki/Samba_(software)) for more details about the project.

[55] See the Wikipedia (http://en.wikipedia.org/wiki/Packet_sniffer) for more technical details.

generated by some third party protocol. Hence, the analysis of these packets arguably does not represent a decompilation activity.

2.2. Project using (also) decompilation

Some open source projects make extensive use of decompilation; among these projects, we can find those trying to port on Linux device drivers for undocumented, or very poorly documented, hardware designed for Windows. In these cases, reverse engineering the software driver designed for Windows, in order to understand how a similar driver could be realized for Linux, may be easier than directly analyzing the piece of hardware that one wants to make compatible with Linux.[56] Alternatively, the software driver for Windows may be analyzed in order to understand how to create a software "compatibility layer" making the very same driver able to work under Linux.[57] These are probably the only projects doing massive use of decompilation. That having been said, there are several open source projects related to cloning Windows APIs in different ways and they do use various forms of reverse engineering, including decompilation as a last resort tool. The most well-known of these projects is surely Wine, but also ReactOS is of great interest for the paper at hand.

2.2.1. Wine

The Wine project aims at allowing Linux and other Unix-like systems to directly run applications designed for Windows, ideally without the user noticing any difference with respect to Unix-native applications. The name "Wine" derives from the recursive acronym[58] "Wine Is Not an Emulator" and it stresses the fact that Wine is indeed not an emulator or a virtual machine on which a copy of Windows is installed and/or simulated.[59] Instead, Wine implements a "compatibility layer" on Unix-like systems; *id est*, it re-implements on these systems alternative versions of Windows APIs and DLLs, which are called by Windows' applications. In other words, the project re-implements Windows interfaces as if they were enriching the set of interfaces already available on Linux and similar systems. Moreover, "Wine also provides a software library known as Winelib, [that] developers can compile Windows applications against[60] to help port them to Unix-like systems."[61] That means that even software packages, which are not able to directly run under Linux, thanks to

---

[56] In other words, a driver could be decompiled in order to generate a new (typically, open source) driver, which is compatible with the original one. In principle, this is an issue of horizontal interoperability (since a software is decompiled in order to gain interoperability with something that is interoperable with that software, and not directly with it). However, I mention this issue here, speaking about drivers, and I want to explicitly signal it, because it gives empirical significance to a theoretical legal issue, concerning the interpretation of article 6 of the European Software Directive: is it legitimate to decompile a software not to achieve interoperability with it or with another software, but just to achieve interoperability with existing hardware? The wording of the directive is not immediately clear about that. However, the Preamble of the Directive clarifies that (1) "term 'computer program' shall include programs in any form, including those which are incorporated into hardware", (2) the concept of interoperability includes interoperability with hardware components ("interconnection and interaction between elements of software and hardware") and that (3) the purpose of the decompilation exception "is to make it possible to connect all components of a computer system [including both hardware and software], including those of different manufacturers, so that they can work together". For these reasons, I consider that the purpose of achieving interoperability with an existing piece of hardware is a legitimate goal under the Software Directive. However, this interpretation is far from providing the optimal degree of legal certainty that a reverse engineer would like.

[57] It may be interesting to notice that this approach (the one of passing through a software "compatibility layer") transforms a horizontal interoperability problems (i.e. realizing a driver for Linux substituting the existing driver for Windows, making interoperability possible between a given piece of hardware and an operating system) into a vertical interoperability problem between two pieces of software (the software driver for Windows and Linux).

[58] Recursive acronyms are very popular in informatics in general and in the open source community in particular: another famous example is GNU (GNU's Not Unix). See http://en.wikipedia.org/wiki/Recursive_acronyms (last visited July 4, 2008).

[59] "Wine does not require Microsoft Windows" to be installed on the same computer where the Unix system running Wine resides, "however Wine can optionally use native Windows DLLs if they are available", hence optionally doing something which is similar to what (the cheapest version of) OS/2 or Wabi did. (See http://www.winehq.org/, "About Wine".) To be sure, that can be done only if the user running Wine also has a legitimate installation of Windows on the hard disk of his or her PC. However, anybody that ever bought a PC knows how difficult it is to find a personal computer for sale without an original copy of Windows pre-installed, hence this (technical and legal) pre-condition is frequently respected.

[60] Compiling an application against a library means that a given library is used during the "compilation process" transforming the humanly written source code into directly executable object code. In this case, for instance, libraries that were Windows-specific could make the compilation process impossible (or otherwise problematic) under Linux: Winelib solves or eases similar problems, making it easier to realize object code that is executable under Linux or other Unix-like operating systems.

[61] *See* the Wikipedia, http://en.wikipedia.org/wiki/Wine_(software). See also the document "Debunking Wine Myths" on Wine's official website (available at http://www.winehq.org/site/myths; last visited July 31, 2008). There it is clarified that "APIs are like a library – it's always nice to have as many books on the shelves as possible, but in reality a few select books are referenced over and over again. Most applications are written to the lowest common denominator in order to have the widest audience possible.

Wine (and about half of existing Windows applications still do not) could easily be "ported" to Linux (at a fraction of the cost that would normally be necessary to bear) by their own developers (or by other developers having access to the original source code).

As stressed at the top of Wine's official homepage,[62] Wine is a complete reimplementation of Windows's API. That means it is not violating Microsoft's copyright, at least as long as the thesis of the first paper of this dissertation holds and as long as what Wine's developers claim is true: "Wine […] is a completely free alternative implementation of the Windows API consisting of 100% non-Microsoft code".

Even though Wine is still under intense development (despite the release of version 1.0), it already runs several important applications[63], like Microsoft Word, Excel and Power Point from various major releases of Office (including XP, 2003 and 2007) or other "killer applications", like Adobe's Photoshop CS2 or some versions of AutoCAD and several PC games. And, what is probably more relevant, it can run several simple custom applications, created ad hoc for the needs of a variety of Windows business users, starting from Windows 3.1, 95 and 98. Of course, my intention is not to advertize the Wine project: I detail these results because they show how significant a similar project may be in lowering the aforementioned application barrier to entry, which may prevent Unix-like systems from representing a credible competitive threat to the dominant Windows platform. In fact, Wine developers are perfectly aware of this role of their software. They explicitly refer to the "chicken and egg" problem described above using the words of Judge Jackson and stress the importance of the multitude of more or less significant sunk costs, creating a huge inertia as far as the leading operating systems are concerned:

> The dependency is not so much on Microsoft Windows as it is on Windows applications. Boxed off-the-shelf applications, games, in-house applications, vertical market applications, are what prevents users, companies and governments from switching to another operating system. Even if 90% of the needs of most users are taken care of if you can provide them with an office suite, an email client, a browser, and a media player, then there will still be a remaining 10% of their needs, potentially critical needs, that are not met. Unfortunately these remaining 10% are spread across a wide spectrum of applications: thousands of applications running the gamut from games to specialized accounting software for French farms, via Italian encyclopedias, German tax software, child education software, banking software, in-house software representing years of development, etc. It is the availability of all this software that makes Windows so compelling and its monopoly so strong. No platform will become mainstream unless it runs a significant portion of that software and lets individuals, companies and governments preserve their investments in that software.[64]

According to the coordinator of the Wine project, on average Wine developers are now "running a couple of years behind Microsoft" in implementing new features[65] (and, to be sure, these implementations concern the officially documented version of Windows' API and not all the small bugs and problems, which sometimes have to be reproduced in order to get applications designed and tested for Microsoft Windows to actually work under Wine). In fact, a crucial and interesting observation concerns bugs. One has to be aware that Wine developers do not want to implement a good set of APIs (they typically assume that Unix systems already have a much cleaner set of APIs than Windows, and in most of the cases, they would prefer improving Unix' APIs instead of the Windows' ones). They just want to run applications designed for Windows under Unix/Linux. Hence, since perfect interoperability with existing applications is the goal, then Windows bugs have to be re-implemented as well, at least if they are used (or somehow dealt with) by existing software.[66] To clarify this problem with an absurd, but easily understandable example, assume that a Windows *SquareRoot*(x, n) API existed and imagine that a bug in Windows returned 3 as the square root of 4,

---

Windows XP support is simply not that important - most applications only require Windows 95 or Windows 98. Currently Wine supports over 90% of the calls found in popular Windows specifications such as ECMA-234 and Open32. Wine is still adding Win32 APIs, but a lot of the work right now involves fixing the existing functions and making architecture changes."

[62] See http://www.winehq.org/, "About Wine".

[63] See the database of supported applications on Wine's website for more details: http://appdb.winehq.org/.

[64] See the document "Why Wine is so important" on Wine's official website (available at http://www.winehq.org/site/why; last visited July 31, 2008).

[65] Interview with Alexandre Juilliard (coordinator of the Wine project and CodeWeavers Chief Technology Officer) on LugRadio (December 2007), available at http://www.temsc.co.uk/lugradio/lugradio-s05e07-171207-high.mp3 (last visited July 31, 2008).

[66] "We try to implement the bugs, or at least the ones that applications depend on. The only reason for implementing the Win32 API is to run all the applications written to it, there is no point in trying to improve on it if it breaks compatibility." Alexandre Julliard in Eugenia Loli-Queru, *Interview with WINE's Alexandre Julliard*, on OSNews.com, October 21, 2001 (available at http://www.osnews.com/story/227; last visited June 20, 2008).

with all other square roots returned correctly. Then, assume that all applications needing to evaluate a square root dealt with this issue subtracting 1 to the result returned by the appropriate Windows API, when the argument is 4. Obviously, this API specification would be something like "*SquareRoot*(x, n) returns the square root of x, with n digits of precision". However, if Wine correctly implemented the API and did not replicate the Windows' bug, then all the applications developed and tested under Windows would receive the correct result of 2 from Wine's API, but then they would subtract 1 (in order to "correct" the expected bug), so that the end user would get 1 as the square root of 4. Of course, such an absurd bug would have been corrected under Windows in the first place; however, similar (but much more complex) cases do exist and applications could be "broken" by correcting strange behaviors of APIs that do not conform to known specifications. If Wine has to be compatible with Windows, it has to conform with the actual specification implemented by Microsoft (in our example: "*SquareRoot*(x, n) returns the square root of x, with n digits of precision, apart from the case x=4, where the result is always 3") and not with the one published by Microsoft for developers of complementary applications.

Coming back to the issue of lead-time, even assuming that Wine is able to keep pace with Microsoft in implementing new APIs, additional delays may result from the fact that major technological shifts cannot be dealt with by Wine, until they are actually released by Microsoft. (Notice that, on the contrary, Microsoft engineers may work on these APIs months or years before finally releasing them, and in some cases even before announcing them.) For instance, in 1998 Wine achieved "pretty good Win16 support",[67] which achieved a good compatibility with Windows 3.1, however, in the meantime, Microsoft had released Windows 95 and was releasing Windows 98, which were running also 32-bit applications (a major technological shift, responding to the availability of a new category of microprocessors). Today something similar is likely to happen with the release of Windows XP and Windows Vista 64-bit, which are not yet supported by Wine, despite the good result that this piece of software – after ten year of intense development – has been able to reach in the field of 32-bit compatibility. At the moment, there are almost no applications for Windows available only in 64-bit version, but if Microsoft were able to increase the speed of innovation, Wine and similar projects would likely lag behind for a quite some time.[68] Hence, Wines may lag a couple of years behind Microsoft when normal incremental innovation is taking place, but potentially much more (even five or ten years, as happened with the shift from the 16 to the 32-bit technology) when more radical technological shifts are happening.

Studying how the Wine project works also dispels a kind of "urban myth" that found its way into academic thinking as well. Actual software decompilation is pretty rare and it is typically not the way Wine developers proceed, since it is very complex and time-consuming and because, in order to achieve the much needed vertical interoperability, Microsoft itself documented the vast majority of the most useful Windows APIs. In fact, even though Microsoft's documentation of Windows APIs may frequently be poor, the way Wine developers proceed in re-implementing the specification is the following:

> Usually we start from whatever documentation is available, implement a first version of the function, and then as we find problems with applications that call this function we fix the behavior until it is what the application expects, which is usually quite far from what the documentation states.
> This is of course a time-consuming process, since a single problem can have multiple causes and it's not always easy to find which function needs fixing. We have a number of tracing and debugging tools to help, and beyond that we depend on having a lot of people test a large variety of application[s]; this is one of the big advantages of the open source development model.[69]

Obviously, understanding how applications designed for Windows fail to work, under a given reimplementation of Windows APIs, *is* a form of reverse engineering, but it is not decompilation. That confirms that decompilation is really a solution of last resort. Not only could an extensive use of reverse engineering attract most unwanted attention from the legal point of view and could offer to Microsoft the possibility of arguing that Wine violates its copyright (a problem that I will address later on), but also the cost and complexity of decompilation should be considered. In fact, if Wine is able to more or less keep pace with Microsoft in implementing new APIs, it is also because the majority of these APIs are normally quite well

[67] See "WWN Wine 1.0 Interview Series! Interview: Alexandre Julliard", available at http://www.winehq.org/?issue=348.

[68] See "WWN Wine 1.0 Interview Series! Interview: Alexandre Julliard", available at http://www.winehq.org/?issue=348: "In the longer run, 64-bit support will clearly require some deep technical changes; it's fortunately not as bad as the 16->32 transition, but it's still a lot of work."

[69] Alexandre Julliard in Eugenia Loli-Queru, *Interview with WINE's Alexandre Julliard*, on OSNews.com, October 21, 2001 (available at http://www.osnews.com/story/227; last visited June 20, 2008).

(even if not optimally) documented. That is the case because, for Microsoft, deciding not to document new APIs would entail an enormous cost in terms of lacked opportunities of fostering the creation of new applications, which are complementary to Windows (hence, foregoing an increase in the value of the operating system). For this reason, Microsoft probably evaluates that the cost of excessive secrecy (in terms of reduced vertical interoperability) would be higher than its benefits (in terms of reduced horizontal interoperability). If this were not the case (as for some specific APIs or applications, where Microsoft wants to keep tight control also on vertical interoperability), the costs of a project like Wine would increase significantly and software decompilation would be an even more crucial tool to resort to.

To complete the description of the Wine project, notice that several commercial projects build on Wine in order to offer professional support, improved compatibility with some kind of applications and – in particular – a much easier installation process (of the software itself, but – in particular – of the various Windows applications supported), needing no or just very limited intervention from the user. The main example is CrossOver from CodeWeavers, which is also the "leading corporate backer of the Wine Project".[70] Another example is TransGaming's Cedega,[71] which is focused on making last generation games available under Linux. Yet another example is Bordeaux[72]. These projects do not simply act as parasites of the Wine project; to the opposite, they may finance it, employ some of Wine's developers and/or contribute back some of the code they develop to improve Wine. In this way, even though ordinary final users may need to buy these commercial software in order to take advantage of some newly implemented functions, these improvements percolate back into Wine.[73] Furthermore, several open source projects share code – or, at least, reconstructed APIs specifications and similar information – with Wine. An example is the ReactOS project[74] (described below).

Before moving on to the description of other case studies, a final question could be touched upon. If Wine is so useful to reduce the level of the application barrier to entry and if it can help users to migrate to Linux with lower costs, people with some familiarity with Linux may ask themselves why is Wine not so well-known, not only to the average computer user, but also to Linux users. More specifically, why do major Linux distributions not bundle Wine with their standard installation (at least in cases where the installer detects a Windows installation on the same PC, so that it is very likely that Wine would be useful to the user)? According to Wine's project leader:

> I doubt that [major Linux distributions] will bundle Wine, since they are all scared of Microsoft. It's all FUD, there's no rational reason to be afraid, but even if the likelihood of a lawsuit is very small, the infinite resources that Microsoft could put into it is enough to discourage them.[75]

Actually, one should also notice that Novell, a major commercial distributor of Linux, did start bundling Wine in its SUSE Linux over 10 years ago.[76] However, this does not dispel the doubt that the reason for which some Linux distributions do not bundle Wine is the fear of legal actions. Quite to the opposite, Novel is well-known in the open source community for having signed an agreement with Microsoft, precisely shielding it from some potential intellectual property violations;[77] moreover, also Novel does not bundle Wine with its commercial Linux distribution, likely to prevent excessive attrition with Microsoft.[78]

2.2.2. ReactOS and TinyKRNL

If Wine may be perceived as being very (and maybe overbroadly) ambitious, ReactOS is even more so. In fact, this project is trying to create an entire operating system, completely cloning Windows APIs (and, obviously, without copying the original Microsoft code). If completely successful, the project would create a platform on which applications and device drivers written for Microsoft Windows would run as in their native environment. The project started in 1996 and plans to be "suitable for every day use" (even though still

---

[70] See http://www.codeweavers.com/about/.

[71] See http://www.transgaming.com/products/cedega/.

[72] See http://bordeauxgroup.com/.

[73] See also Eugenia Loli-Queru, *Interview with WINE's Alexandre Julliard*, on OSNews.com, October 21, 2001 (available at http://www.osnews.com/story/227; last visited June 20, 2008).

[74] See http://www.reactos.org/en/about_userfaq.html (last visited August 1, 2008).

[75] See "WWN Wine 1.0 Interview Series! Interview: Alexandre Julliard", available at http://www.winehq.org/?issue=348.

[76] See "WWN Interview Series: Marcus Meissner", available at http://www.winehq.org/?issue=347.

[77] See the first paper of the dissertation at hand, § 9.2.1.

[78] See "WWN Interview Series: Marcus Meissner", available at http://www.winehq.org/?issue=347: "For Enterprise products, yes there is a problem of lawsuits which might make most of the distributors afraid I guess."

in beta version, and arguably more suitable for developers and hacker than for the general public) during 2008[79].

The relationship between Wine and ReactOS is described in the following way on the official website of the latter:

> [W]e work very closely with the Wine project. Wine probably has a lot more in common with ReactOS than with Linux. The Wine project has the goal of implementing the entire windows API on top of WineServer. There are only a few WINE dlls that cannot be used in ReactOS. […] The rest of WINE's DLLs can be shared with ReactOS. We have several developers in both the WINE and ReactOS projects that work on cross-compatibility issues between the two projects. It is our view that Linux + Wine can never be a full replacement for Microsoft[R] Windows[R]. ReactOS has the potential for a much higher degree of compatibility - especially for Microsoft[R] Windows[R] drivers - which WINE does not address.[80]

On the project's website[81] it is also possible to find a reference[82] to another project, TinyKRNL. TinyKRNL was (indeed, the project has been abandoned)[83] a much less ambitious project than ReactOS and it aims at the creation of replacements for some specific modules of Windows (focusing on a specific version: Windows 2003 SP 1) for educational and documentation purposes. According to ReactOS website, the cooperation with TinyKRNL must be kept at a very low level, without sharing source code[84], because the methods used to develop TinyKRNL include dirty reverse engineering. Moreover, and what is legally most problematic, TinyKRNL developers consider acceptable any way of achieving a 100% compatible result, including the recreation of source code which is completely identical to the original Microsoft one (even if achieving this would not be technically easy).[85] Apparently, TinyKRNL developers regard their copying as protected by fair use, and this (disputable) opinion is surely influenced by the educational and non-commercial purpose of their activity. However, there is a specific field in which the ReactOS project (a potentially commercial project and wanna-be competitor of Microsoft) do use in an active way the reverse engineering activity performed by TinyKRNL, and this is the field of interfaces. As I argued in the first paper of this dissertation project, in this field (and with all the qualifications already discussed) it is possible to reproduce the "external part" of an interface, since doing so is necessary for technical purposes in order to achieve interoperability. Hence, it is reasonable to argue that ReactOS developers are free to learn as much as they like from TinyKRNL interfaces, as long as the source code implementing the "black box" part of these interfaces (i.e. the part of the interface that needs not to be identical to the original one for compatibility purposes) is independently written from scratch[86]. Indeed, ReactOS developers seem to be perfectly aware that they have to create a "clean implementation" of the APIs and similar functions, the working of which they may – in some cases – understand also thanks to the "dirty reverse engineering" performed by TinyKRNL or similar projects:

> [T]he great thing is that TinyKRNL will provide the most complete documentation of the most recent and technically advanced version of a released NT-family operating system – Windows 2003 SP1. ReactOS developers can use this documentation for reference when creating a clean implementation of functions or improving already developed code.

---

[79] See http://www.reactos.org/en/about_roadmap.html (last visited August 1, 2008).

[80] See http://www.reactos.org/en/dev_faq.html (last visited August 1, 2008).

[81] See http://www.reactos.org.

[82] See http://www.reactos.org/wiki/index.php/TinyKRNL (last visited September 14th 2007).

[83] A copy of TinyKRNL website is still available here: http://web.archive.org/web/20070818064549/http://tinykrnl.org/ (last visited August 4, 2008).

[84] Ibid. "Unfortunately, due to copyright laws and other law-related stuff, ReactOS (which aims at commercial usage too) can not directly utilize methods of development like dirty reverse engineering, and thus ReactOS can not share all code with the TinyKRNL project like we are sharing code with WINE."

[85] Ibid. "The methods used for development of TinyKRNL's modules source code involve all possible methods of achieving the end result of having a 100% compatible (or even identical) result. Reverse engineering is one of them (mainly so-called 'dirty' way)".

[86] To be sure, the fact that some competitors of Microsoft product (the decompiled software producer) are clearly allowed to learn from TinyKRNL activity would likely be taken into account by the majority of courts in the fair use test concerning TinyKRNL project itself. In fact, this situation is different from the typical decompilation fair use (that I discussed in the first paper composing this dissertation). On the one hand, in that setting, the decompiled code was not distributed to third parties (but used only by decompilers themselves to learn about a software) – at most, it was a technical description of the interface that could be shared. On the other hand, TinyKRNL is (at least according to ReactOS website) copying also the expression of Microsoft's source code in cases in which this copying is not necessary for technical purposes (the fact that there may be educational purposes should be taken into account, but cannot offer a complete shield against violations).

Notice that – as it frequently happens – ReactOS developers seem to be quite concerned about US copyright law (and, in fact, they care about clean and dirty-room reverse engineering, that US courts sometimes consider as a discriminant between lawful and unlawful projects), but appear to be completely unaware of (and uninterested in) the European legal setting (but I will discuss more about that later on). That is even clearer for TinyKRNL developers. In general, one may argue, TinyKRNL's choice of using not only dirty reverse engineering, but of distributing also a derivative work (i.e. the code directly re-created starting from reverse engineering), is a copyright infringement. In fact, it is not at all clear that projects like TinyKRNL may enjoy a fair use exception, not even considering that they are working on an educational project and that they are not creating a competing product with respect to the decompiled one. What is more, TinyKRNL developers may or not enjoy a fair use exception, but – one could argue – they are certainly violating Article 6 of the European Software Directive in more than one way (see below, § *4. Decompilation in the EU*).

In any case, the ReactOS developers seem to have recently taken very seriously the risk that they may be violating US copyright law. In a letter to the other contributors, Steven Edwards - ReactOS and Wine developer, wrote:[87]

> For us in the US when you speak of clean-room reverse engineering it means that one person tears apart the implementation of a device, writes documentation and another reads that documentation and implements. Other countries do not require this invisible great wall of development and allow the same person that disassembles the interface to also write the replacement implementation.

This letter offers a correct (even thought quite rough) description of the clean room reverse engineering practice, but it assumes that this practice is a legal requirement in the US (and not in some other countries). (In fact, the clean-room reverse engineering process has several legal advantages in the US, as I will discuss, but stating that this is a strict legal requirement is ultimately incorrect.) In addition, discussions on the mailing list of ReactOS offer some clear examples of the uncertainty and fear spread among developers.[88] Of course, in response to the uncertainty and doubts concerning the legal status of software reverse engineering (in particular in the US), ReactOS developers actually started to adopt quite prudent policies.[89] Issues tackled included: (1) amending their Intellectual Property Policy Statement mandating clean-room reverse engineering; (2) auditing the entire source code, with the goal of rewriting all the code obtained through "dirty" reverse engineering or the origin of which cannot be traced to publicly available specifications or other surely legitimate sources; (3) requiring developers to accept in writing the aforementioned IP Policy Document; (4) to avoid any doubt, preventing developers having had access to potential Microsoft's trade secrets in certain fields from contributing code in these fields.[90]

At least two comments about these policies can be anticipated here (and their relevance will be clearer in what follows). On the one hand, all these policies are surely appropriate in order to grant to ReactOS a better

---

[87] Steven Edwards, *Reset, Reboot, Restart, legal issues and the long road to 0.3*, 2006-01-27, available at http://www.reactos.org/pl/news_page_14.html (last visited July 20, 2008).

[88] Archive available at http://www.reactos.org/pipermail/ros-kernel/. For instance, a developer argued that the 9th Circuit had established the legitimacy of software reverse engineering and decompilation, at least in order to achieve interoperability, in the Sega and Connectix cases: see Casper Hornstrup's messages, May 2004, available at http://www.reactos.org/pipermail/ros-kernel/2004-May/003667.html: "Where Microsoft does not document their OS, disassembly is the only way for us to 'gain access to the ideas and functional elements embodied in a copyrighted computer program'. […] I believe cloning the OS is a 'legitimate reason for seeking such access'. Connectix's reason was 'building an emulator'. Now, of course, you cannot just copy the assembly instructions needed to implement the API, but you can use what you have learned from the disassembly to express the implementation of the API so it behaves identical." In response, another commentator replied that "The 9th Circuit court is called the 9th Circus in the US because It is the most overturned court in the world. Just because they hand down a judgement does not make it the law of the land in the US. Only in those states that are under the [jurisdiction] of the 9th Circus. The rest of the county is still free to [enforce] the law as they see it until the Supreme Court steps in and overturns or agrees with one of the lower courts." See Steven Edwards' messages, May 2004, http://www.reactos.org/pipermail/ros-kernel/2004-May/003675.html. The discussion more or less stopped with people divided on similar positions, with the majority of developers being clearly unsure about the actual legal status of decompilation in the US.

[89] Steven Edwards, *Reset, Reboot, Restart, legal issues and the long road to 0.3*, 2006-01-27, available at http://www.reactos.org/pl/news_page_14.html (last visited July 20, 2008).

[90] Essentially, the Audit process will consist in "commit[ing] all documentation [of] reverse engineer[ing activity], so that someone else can reimplement it." The tools to collect these pieces of information (essentially reconstructed API specifications and similar documents) are websites, wikis and other quite open and easily accessible workgroup platforms used by developers. Obviously, code which has been created re-implementing specifications found "on MSDN, Google, sysinternals, osronline, any book published by Microsoft Press or any other publication" will be considered "clean". See http://www.reactos.org/wiki/index.php/Audit.

chance of preventing legal problems and eventually to solve them, in case of lawsuits. However, as I will show, clean-room reverse engineering should be interpreted as a useful precaution and not as a necessary precondition to perform software decompilation to achieve interoperability. On the other hand, if part of the development of ReactOS is performed in Europe, developers should also be more careful in respecting article 6 of the Software Directive. In fact, there are reasons to think that the specific policy to guarantee clean room reverse engineering which is being proposed at ReactOS violates some of the conditions that must be in place in order to enjoy the protection offered by article 6, at least unless some precautions are taken.

3. The simple economics of decompilation

In the next paragraphs, I will discuss the conditions that make economically sustainable the creation and development of new pieces of software, in a setting in which functional equivalence and perfect interoperability may be achieved through decompilation. In other words, I will show in which economic and legal setting the sunk costs necessary to write a new piece of software could be recouped, taking into account the competitive pressure represented by decompilation.

Before going on, a preliminary comment is appropriate.[91] I think that it is, in general, safe to assume that legal systems have the goal of guaranteeing a dynamic and innovative software market. However, there may be various ways to achieve this goal. On the one hand, we may follow the approach I tried to describe in the first paper of this dissertation, where copyright is essentially a tool preventing pure free riding and where the combination of intellectual property and trade secret is designed in such a way as to *guarantee just that – on average – welfare increasing investments are economically sustainable*. In such a setting, copyright works as a pure property right, but its scope is limited to the expression of ideas; instead, ideas themselves are free to be taken, but possibly protected by trade secrets, working as an implicit liability rule.[92] This approach tries to balance the need for incentives to create in the first place and the need to allow incremental innovation, without creating excessive transaction costs, coming from a plethora of cross veto powers related to excessively broad intellectual property rights. In what follows, I will try to confirm that this is the appropriate path to follow and that trade secret and rules concerning software decompilation may be designed in such a way as to allow the possibility of creating interoperable systems, at the same time preventing excessive free riding and market failures. Firms will innovate whenever performing an investment in innovation improves the competitive position of the undertaking more than not innovating. To integrate such a condition, a certain degree of appropriability for innovations is necessary, but it is also necessary that not-innovating incorporates a significant risk of being caught-up by competitors. Hence, intellectual property should exist, but it should not offer an excessively safe protection against potential competitors.[93] On the other hand, we may cede to the temptation of asking for other conditions *guaranteeing that every socially beneficial investment is performed*. A sufficient condition to achieve that, in particular, would require that the person carrying over an investment receives the entire social benefit generated by his/her efforts. The main problem entailed by this second approach is that making the transfer of this wealth possible could – at a certain point – entail higher transaction costs (and dead weight losses) than the generated benefits in the form of increased incentives. Here, multiple phenomena may play a role. A first, obvious, problem is that generating incentives to innovate through property rules may generate the usual inefficiencies of monopoly, which may be magnified in settings characterized by incremental innovation and high dynamism: reducing output is a negative side effect of monopoly, but it entails higher costs when this output is also the input for further innovations. In other words, if it is true that the pace of innovation is "exponential" in software markets, also the cost of missed

---

[91] This comment concerns my understanding of the economics of intellectual property in general: I remand to the General Introduction of this dissertation project, for additional remarks on this subject.

[92] I will discuss more about that. In brief, if one wants to discover a secret, one may either pay the original developer a fee or invest in a reverse engineering project. In fact, the possibility of "self help" through reverse engineering imposes an "implicit cap" on the cost of licensing the trade secret (equal to the cost of reverse engineering). In such a setting, a refusal to deal of the original developer would not completely impede access to innovation – as in the case of a patent or any property rights that may be used *erga omnes* – but would just force third parties to actually perform reverse engineering.

[93] As Baker put it, using the language of the software industry, the fact that "the social returns to innovation exceed the private returns […] is a 'feature' of competition, not a 'bug.' Contrary to what is sometimes suggested, this observation does not imply that the key to more innovation is to allow firms to appropriate more of the social benefits of their new products and production processes". Indeed, investments in innovation will be performer as long as "the incentive to escape current product market competition […] is more powerful than the fear of post-innovation product market competition […] in the decision-making calculus of potential innovators." See JONATHAN B. BAKER, *Beyond Schumpeter vs. Arrow: How Antitrust Fosters Innovation* (June, 2007), p. 8—9. See also the General Introduction, § *2.1.2. Models of Competition: Schumpeter vs. Arrow.*

innovations grows much more than linearly. Moreover, transaction costs related to the existence of strong property rights risk being, in a large proportion, external costs, from the original investor's point of view, so that he/she may want to collect these benefits also in cases in which collecting them is suboptimal.[94] In general, we may face a situation where transaction costs prevent the Coase theorem from actually applying: in other words, creating strong and well defined property rules on any kind of innovation generated in software markets may not be an optimal solution, simply because parties will not perform several welfare increasing bargains because of the existence of transaction costs. Additionally, not only ordinary transaction costs may be at place, but also market failures generated by strategic reasons and/or asymmetries of information.[95] For instance, in a setting dominated by uncertainty, a rightholder may prefer to plan the obsolescence of its own product, instead of entering complex cross license agreements with other entrepreneurs, which think to be able to improve existing products covered by intellectual property rights.

3.1. The simple economics of decompilation before open source

Having explicitly clarified that the existence of transaction costs makes it normally impossible and frequently undesirable to have "perfect incentives to create", and also that property rules could be "abused" for strategic reasons, or simply fail to be socially optimal because of other market failures, we may take an additional step. To do so, I will decline, in the context of software, what some scholars have argued about reverse engineering and trade secret in general:

> as long as the task of reverse engineering proves neither too hard nor too easy, investment strategies affecting the pace and direction of innovation that result from […] individual assessments of market prospects are as procompetitive as the free-market system permits.[96]

In other words, there is an interval for the cost of reverse engineering, in which new entry decisions do not create market failures; instead, they just put some competitive pressure to innovate on incumbents, which are still able to recoup their investments in innovation. Of course, that is true just as long as "reverse engineering proves neither too hard nor too easy". To define what is "too hard" and what is "too easy" we need to be more precise and make some assumptions, also taking into account uncertainty. For the moment, I will assume that the level of uncertainty about the market prospects for products embedding a given innovation is fairly similar for both the first comer and any given late comer. This is clearly a simplifying assumption that allows for the ignoring of uncertainty in the following discussion. However, I will discuss later on that – as a first approximation – this assumption is less unrealistic than one might imagine.

Obviously, no IPR can guarantee the recoupment of sunk costs: this depends on the market response to the product (and any alternative would just incentivize waste). However, if the product is successful enough to guarantee recoupment at least in case of complete monopoly, I argue that new entries will not be a major obstacle to this recoupment, as long as the sunk costs for new entrants will not be significantly lower than the sunk costs for the first entrant,[97] i.e. as long as barriers to entry are not negatives.[98] This condition can be formalized in this way:

$$[\text{sunk (entry) costs of late entrant}] \geq [\text{sunk (R\&D) costs of first entrant}]$$

Actually, the socially optimal situation is the one in which the previous condition holds with equality and there is a potentially infinite supply of new entrants. In general, entrants will keep entering as long as their expected profits are equal to their sunk costs (not yet sunk at the moment in which they evaluate the opportunity to enter the market). If these entry costs are as high as the R&D costs of the incumbent, then no market failure will result, but – at the same time – the incumbent will be prevented from extracting a monopolistic rent from the market (actually, doing so will be possible only during the period of time needed

---

[94] For instance, copyright and patent holders do not fully sustain the costs associated with the administration and enforcement of the copyright or patent system. This problem may be magnified if supra-compensatory damages are granted to intellectual property owners.

[95] I will not discuss here the conditions at which the controller of a platform (frequently exercising this control also through intellectual property rights) is likely to abuse of its position. I will discuss some of these issues in the third paper of the dissertation; for the moment I just remand to the taxonomy of these situations provided by the ICE paradigm, developed by Farrell and Weiser. *See* FARRELL & WEISER, *Modularity, Vertical Integration, and Open Access*, .

[96] *See* REICHMAN, *Legal Hybrids*, , p. 2530. See also, among others, PAMULE SAMUELSON & S. SCOTCHMER, *The Law and Economics of Reverse Engineering*, 111 Yale Law Journal, 1575--1663 (2002).

[97] In fact, fixed costs can be safely ignored, as long as they are the same for everybody. What I will try to analyse are precisely the conditions under which these fixed costs are roughly the same for every competitor.

[98] See the General Introduction, § *2.1 Cost structure*.

to late comers in order to build their production capacity, i.e. to develop and test their competing products). Clearly, in this description I made use of the assumption concerning the absence of risk (*rectius*, the presence of symmetric risks between the incumbent and the new entrants). In the extreme case in which the incumbent faces all the risk and late entrants none, to have technological advance in the first place we need the profit made during the lead time of the incumbent to entirely compensate for the risk taken.

A similar reasoning, concerning the lead-time of the first entrant, applies also to cases in which late comers may enjoy some free riding. In fact, we may expect that the first entrant would still invest in creating a new piece of software even if he/she expects the competitors to have slightly lower costs of entry in the future. In particular, barriers to entry may be negative and the market could still work properly, as long as the lead time of the first comer is sufficient to generate revenues that compensate the reduction in the cost of entry.[99] In evaluating the length of lead-time, consider that software development is still a labor-intensive industry and the supply of skilled developers is far from infinite: late comers will need time and resources to convince programmers to work with them and/or train them (and that is a particularly significant problem, in case they are financially weaker than the incumbent). For instance, when asked what he would do if he had a "big bucket of money", Wine's lead developer Jeremy White replied:

> [T]he reality is that a big bucket of money wouldn't solve the biggest problem – having talented Wine developers. It might help, but good Wine developers are very rare, and I don't know that we can force them into existence.[100]

We should also notice that, depending on the kind of competition that they may expect on the market (i.e., with almost any kind of competition but perfect Bertrand competition with homogeneous products),[101] new entrants will likely enter into profitable markets, even in cases in which their sunk costs are higher than those of the first comer. In fact, the incumbent does not normally have incentives to fiercely fight entrance; instead, he may just accommodate and enjoy a slightly reduced profit.[102]

Overall, a condition that could guarantee a good (even if not necessarily optimal) level of technological development would be the following:

$$\text{sunk costs of each late entrant} \approx \text{sunk costs of first entrant}$$

The fact that this condition is just an approximation should not worry us too much: stricter assumption could allow for writing clearer conditions, but their empirical relevance would be questionable and – after all – the legal tools intellectual property offers in order to fine tune the competitive process will require even bigger approximations!

To say something more, about the previous condition, a promising path may consist in decomposing it in a more analytical way. In particular, if entry requires a reverse engineering project mainly focused on achieving interoperability, using the wording of the first paper of this dissertation, we may restate the previous condition as follows:

$$\text{cost of reconstructing an interface specification (if any)} + \text{cost of reimplementation (i.e., new expression)} \approx \approx \text{cost of research (i.e. devising the original specification)} + \text{cost of original implementation (i. e. original form or expression)}$$

Clearly, the "cost of reconstructing an interface specification" includes all "reverse engineering costs" (if any), which may or not entail disassembly and decompilation. Moreover, it is important to notice, as

---

[99] In other words, no market failure occurs as long as the reduction in the incumbent's earnings due to new entrants is compensated by the profits gathered during the lead-time.

[100] See WWN Interview Series: Jeremy White, available at http://www.winehq.org/?issue=347. Obviously, he also stressed that "a lot of things" could be done with money, for instance "make test to work perfectly on every machine" and other routine activities; however, developing true programming capacity may be limited by the availability of skilled programmers and financial capacity helps just in part, since specific training and various skills have to be built.

[101] If there is Bertrand (price) competition with homogeneous products, the first entry would drive prices to zero and that would preclude the possibility of recouping fixed costs. Obviously this is a theoretical possibility, but similar situations would deter entry. However, situations like the ones described in footnote 102 are much more likely in real markets.

[102] A classic "entry game," solved by backward induction, may show that the incumbent will never have an incentive to fight entry in an excessively strong way, once new entrants are in the market. Of course, introducing a role for reputation and some imperfections in financial markets may radically change this result. In any case, notice that subsequent entrants – i.e. the 3rd and following ones – will not have significant cost advantages over the first of the late comers: when perspective profits will be lower than the cost of entry, entry will cease (assuming away coordination problems for simplicity: formally, one may assume sequential entry choices).

confirmed by the case studies, that new entry is a significant possibility even without recurring to reverse engineering (and to decompilation in particular), because some (more or less accurate) interface specifications may already be available.[103]

Our initial condition may be restated once again, subtracting the cost of actually writing down the code on both sides. In fact, even though the incumbent may have been the first one to devise a certain set of interfaces (and other non protectable ideas and methods), both it and the late comer will have to write down their source code without copying (to avoid copyright violations). Here I stress a distinction between "development" and "research" costs. A similar distinction is proposed also by Guglielmetti[104] (and it is more or less implicit in various other authors writing about software)[105]. The point is that devising the general structure of interfaces may entail some (more or less significant) "research costs", but the fruits of these costs are not protected by copyright. At most, they could be protected by patents, but – as I discussed in the first paper – this is rarely the case (and, here, I assume that it is not). In any case, however, actually writing down a program behaving according to the devised API specification entails significant cost, which I will label "development costs". Hence, if we subtract "development costs on both sides), we obtain the following condition:

(cost of reverse engineering) + (cost of avoiding copyright infringement during or after reverse engineering and reimplementation) ≈ research cost of the incumbent

In fact, I cheated a bit, introducing a "cost of avoiding copyright infringement during or after reverse engineering". That derives from the fact that the "cost of reimplementation" may be different from the "cost of original implementation". In the first case, on the one hand there may be additional costs related to the need to avoid infringement, or the suspect of infringement (auditing costs, etc.); moreover, if a clean room reverse engineering process is performed, just because it is needed in order to avoid the risk of a finding of

---

[103] Clearly, also forgetting interoperability and just proposing the market a new product which is a functional substitute of the original software is a possibility. I will not discuss this case in detail, because it is not related to reverse engineering and also because basic reasons related to freedom of expression and economic initiative suggest me that this situation cannot but be legitimate and even desirable. However, in principle, also this kind of entry could be problematic, if the original developer created – with significant costs – some new technical solution that can nevertheless be easily appropriated just looking at his product. If that happened more than very sporadically, software patents would be needed to protect innovations of this kind. However, I am not aware of similar situations in software markets and I already discussed in the first paper the reasons for which I am very sceptical about granting software patents (and – given the significant pace of innovation of the software industry in a situation of substantial absence of patent protection – I think that the burden of proof concerning the opportunity of this kind of patents should be on the shoulders of proponents).

In fact, a late comer will enter with an interoperable product as long as:

(cost of reverse engineering) + (cost of reimplementation) ≤ (expected revenues from an interoperable substitute)

Since we saw that there is a choice among entering with an interoperable product or with a fully independent one, the "interoperable entry" will be chosen as long as entry is profitable at all and:

(cost of reverse engineering) + (cost of reimplementation) – (expected revenues from an interoperable substitute) ≥ (Revenues from a non-interoperable substitute) – (cost of developing a non-interoperable substitute)

Actually, we are able to do some additional "algebra". In fact, the "cost of reverse engineering" is incurred only as long as one wants an interoperable product, otherwise a simple (and cheap) black box analysis could be performed to understand what kind of functions a given piece of software delivers to users. At first approximation, I will assume that this second kind of analysis is essentially free and neglect it for simplicity (alternatively, you may assume that it takes place in any case and delete them on both sides of the condition). Hence, the cost of reverse engineering is an additional cost relevant only for "interoperable entry".

As far as the "cost of reimplementation" is concerned, notice that it may be different from the "cost of developing a non-interoperable substitute". In the first case, on the one hand there may be additional costs related to the need of avoiding infringement, or the suspect of infringement: if a clean room reverse engineering process is performed, the cost of this process with respect to dirty reverse engineering may be attributed to these "costs of avoiding copyright infringement". On the other hand, it may be possible to free ride on some research costs that are not protected by copyright (the individuation of smart and efficient algorithms, etc.). Apart from that, the cost of developing a new piece of software is likely to be more or less the same of writing down the code for a "non-interoperable entry" (which is also likely to be similar to the development cost of the incumbent, "pure research costs" – i.e. costs related to the development of unprotectable ideas – excluded). That having been said, with some approximations, there will be "interoperable entry" as long as:

(cost of reverse engineering) – (saves on development costs from free riding on ideas due to reverse engineering) + (cost of avoiding copyright infringement) ≤ additional expected revenues from interoperability

I assume that this condition is respected in cases of our interest.

[104] GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè second ed, Milano. 1997), pp. 242—243.

[105] In particular, Id., distinguishes a "phase of inventive research" ("fase della ricerca inventiva") and an "innovation phase", but not generating patentable "inventions" ("fase di innovazione […] non inventiva"). The main point of Guglielmetti is actually that patents should be used as the only tool protecting "inventive research" (as long as the product of this research qualifies for patent protection).

copyright infringement during reimplementation, then the cost of this process with respect to "dirty" reverse engineering may be attribute to these "cost of avoiding copyright infringement". On the other hand, it is true that late comers may free ride on some research costs, the results of which are not protected by copyright or patens (e.g. the individuation of smart and efficient algorithms, etc.). Apart from that, the cost of developing a new piece of software is likely to be more or less the same of writing down the code in the first place.

Specifically for interfaces, the equality becomes:

> (research cost of individuating and reverse engineering pieces of code implementing interfaces and recreating an approximate specification) + (cost of avoiding copyright infringement during or after reverse engineering and reimplementation) ≈ (research cost of the original specification and of the non-copyright-protectable solutions embedded in the original implementation)

Looking at this equality, it is clear that the majority of its elements are technologically determined. However, the legislator could (and according to the thesis of this paper should) make this equality approximately holding, working on the "cost of avoiding copyright infringement during or after reverse engineering". The fact that the previous equation should be held valid by the legislator, that is the fact that subsequent innovators do have to bear a development cost which has to be similar to the one of the first comer (possibly licensing from him the code or otherwise rewriting it), seems to be acknowledged by several legal scholars, however this reasoning frequently remains implicit[106]. To make the equation hold, several policies may be implemented.

On the one hand, when reverse engineering is excessively cheap with respect to the possible free riding on ideas that can be achieved through it, then legal obstacles could be used to artificially reinforce trade secret. Limits on the scope of reverse engineering could be established, so that it is legitimate only at certain conditions and/or only to achieve certain results. For instance, deciding that a clean room process of reverse engineering is formally required, instead of being just a mental experiment conducted by the judge in order to evaluate substantial similarity, could increase the cost of reverse engineering. Or, if reverse engineering is legitimate only to achieve interoperability, it means that the legislator evaluates that the previous equality is respected only for this goal, while reverse engineering to discover ideas and principles not related to interoperability could create market failures. Furthermore, operating directly on the cost of reverse engineering is not strictly necessary. Following the suggestion of Reichman,[107] a blocking period for competing re-implementations could simply be established, in order to avoid market failures, even if the previous equation is not perfectly holding (notice that forcing a certain waiting time before re-implementing may be seen as a special case of increasing the cost of reimplementation).

On the other hand, also excessively increasing the cost of reverse engineering (potentially up to the point of completely banning it) could be socially costly, because it would transform copyright protection of interface implementations in a kind of almost perpetual patent with very low access requirements.[108] In point of fact,

---

[106] A quite explicit reasoning is proposed by Id., . However, the author is actually using this reasoning to argue that patent protection for software needs to be complemented by copyright protection in order to incentive also the "non-inventive phase" of software innovation. "Le condotte che recano un effettivo pregiudizio a tale forma di concorrenza [i.e. all'innovazione non inventiva], e realizzano i rischi in considerazione dei quali [il diritto d'autore sul software] si giustifica , sono quelle che, attraverso la copiatura totale o sostituzioni che possono essere fatte senza costi sostanziali, consentono di trarre un immediato vantaggio dalla maggior vulnerabilità all'imitazione dei programmi per elaboratore rispetto alle altre tecnologie assistite unicamente dal sistema brevettale, impedendo agli innovatori di ottenere la remunerazione dei loro sforzi" (p. 294). Even if I do admit that, in principle, patent protection would be needed in order to provide incentive for finding some applications of "basic software research" (and hence to indirectly provide incentives to basic software research) I already tried to explain in the General Introduction to the dissertation the reasons for which the net effect of granting patent protection to software inventions risks to be socially harmful (see also below, § *6.4.1. Failure of the directive proposal on software* implemented inventions).

[107] REICHMAN, *Legal Hybrids*. See also the *General Introduction* and § *8. Elimination of Free Riding vs. Creation of Economic Monopoly* in the first paper.

[108] If reverse engineering is impossible, the couple formed by copyright and trade secret would create a *de facto* exclusive right with respect to the possibility of creating an interconnection with a given commercial software. In fact, that is true only in a very loose way, or "narrow way", if you prefer, in the sense that the limitation is circumscribed to cases in which decompilation is technically necessary to achieve interoperability. More general technical solutions to technical problems will never receive patent-like protection through copyright: see JAAP H. SPOOR, *Copyright Protection and Reverse Engineering of Software: Implementation and Effects of the EC Directive*, 19 U. Dayton L. Rev., 1063 (1994), p. 1081, according to which "it must be pointed out that to refuse access (through decompilation) to such ideas that are the basis of a computer program is quite another thing than patent protection. It does not in any way form a bar to the working of one's own creativity. A patent will prevent a person who independently develops the same invention from using it in a competing or even in a non-competing program. The harm which the inventor may suffer by disclosing his invention is, therefore, limited. Copyright does not offer any comparable protection to ideas, whether they are novel or not."

there could be cases in which – in the absence of any intervention by law – the cost of reverse engineering is already much higher than the research cost of the first comer. In these cases, simply due to technical reasons, the incumbent would be protected by very high network effects for a potentially long period of time, i.e. until a significant break-through innovation displaces him, without achieving interoperability. In these cases, to have the previous equality holding, it may be appropriate to artificially reduce the cost of reverse engineering.[109] If problems related to the excessive cost of reverse engineering are very severe, a (theoretically) easier way to solve them could be by means of an external intervention on the duration of lead time of incumbents. For instance, one could tailor intellectual property so that the legal fiction protecting compiled (object) code as a literary copyright subject matter would be eliminated – after a certain grace period of time – unless the source code behind it is published as well.[110] However, I do not think that this approach is necessary. At most, a similar, but more limited, solution could be targeted only to interfaces and/or data file formats. In fact, it is only in some fields related to interoperability that the advantage of the incumbent (in term of indirect or direct network effects) could tip the market in its favor. Indeed, in fields different from the achievement of interoperability, equivalent technical results could normally be achieved writing code from scratch without accessing to the original source code, nor to an approximate reconstruction of it. Moreover, the achievement of interoperability is a really necessary step, which public agencies may want to impose, just when this need concerns quasi-monopoly platforms. I will discuss the more about the possible active intervention of public agencies in order to mandate interoperability in the third paper of this dissertation, where I address antitrust concerns.

As a parenthesis, notice that there are cases in which the legislator actually decided to make the "cost of avoiding copyright infringement during or after reverse engineering and reimplementation" negative. In particular, the legal rules governing reverse engineering of semiconductors in order to reconstruct their topographies and mask works allow copying part of the external form ("expression") of the original semiconductors, if a reverse engineering activity has been performed.[111] In other words, not only in this field is there a perfect shield for reverse engineering. But also the fact of performing reverse engineering gives to the agent having performed this analysis, the right to reproduce some of the "external expression" of the analyzed object. In a way, this subsidizes reverse engineering. However, notice that – to enjoy the special legal favor concerning reimplementation that could violate the "quasi-copyright" on the reverse engineering semiconductor – also the late comer is required to produce something "original", i.e. to engage also in some significant "forward engineering".[112]

Before making this informal model of reverse engineering slightly more complex, let me also stress that my description overlooked a potentially significant tool that the incumbent could use to (at least partially) recoup its sunk costs. In fact – if there is a credible reverse engineering project – platform controllers should rationally license their API specifications, instead of waiting for competitors to use reverse engineering to achieve the same result. And this may be good for platform controllers, who can earn some return, but it is also good for social welfare, because we have a simple pecuniary transfer, instead of the a social waste of resources coming from reverse engineering (rediscovering things which had already been developed). Notice that, in principle, also APIs implementations could be licensed, if the threat of independently and effectively re-implementing them is credible.[113]

---

[109] That may be done in various way, for instance giving fiscal advantages to firms practicing decompilation and/or giving decompilation expenditures an especially favourable treatment as research and development investments. To be sure, I am not suggesting this specific policy: it is just an example showing that "negative" legal barriers could be created, i.e. legal incentives to decompile. Another, more realistic even though specific, example is shown below. See f.n. 111 and 112 and the accompanying text.

[110] Something similar has been proposed, for instance, in PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, 94 Columbia Law Review, 2308--2431 (1994).

[111] See the US Semiconductor Chip Protection Act of 1984 (SCPA); and Council Directive 87/54/EEC of 16 December 1986 on the legal protection of topographies of semiconductor products. See also SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*, 1595 ff.; and GIOVANNI GUGLIELMETTI, *Le topografie dei semiconduttori*, AIDA, 191 (1992).

[112] See also SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*: "The Semiconductor Chip Protection Act […] permits intermediate copying of chip circuitry for purposes of study and analysis; it also permits reuse of some know-how discerned in the reverse engineering process. This is a useful boost to competitors designing integrated circuits. The SCPA, however, requires reverse engineers to design an 'original' chip rather than simply making a clone or near-clone of the integrated circuit that was reverse-engineered."

[113] However, the threat may not be credible and – in any case – the leader could be "risk lover" in this field (for the incumbent, we are on the loss side of the game) or he could simply be confident that he will be able to develop new functions before the end of the reverse engineering process and that the follower will always be trying to catch up (while agreement not to use the licensed APIs for an equivalent amount of time could be difficult to enforce and/or risk antitrust scrutiny).

3.1.1. New entrants after the first one

Typically, the reasoning above should apply to any new entrant, not only to the first one. In fact, any new late comer will have to reverse engineer, from scratch, the original piece of software (with whom it wants to achieve interoperability). Indeed, the source code of other late comers will normally be as secret as that of the original incumbent, because the first-late-comer to reverse engineer will be willing to maintain its competitive advantage over other-potential-entrants (exactly as the incumbent was willing to exclude him).[114] Overall, I see no reasons to think that the first (or other) entrants should have higher incentives to license their API (specifications and implementations) than the incumbent.[115] In any case, the possibility that potential new entrants license this information to other late comers should not particularly worry incumbents. In fact, as long as entrants want to recoup their sunk costs and these costs are similar to those of the incumbent, they will not trigger excessive entry.

In the medium/long run, the increasing competitive pressure in the market will progressively reduce profits, up to the point in which new entrants do not have any incentive to support sunk entry costs. However – apart from coordination problems that we may assume away at first approximation – new entries will stop before the revenues for potential entrants drop under the sunk cost of entry; and (if the structure of costs of the incumbent and of the potential entrants are similar) this should not create market failures.

3.2. Evidence concerning the cost of software reverse engineering

As shown above, many of the problems, concerning the opportunity of allowing software reverse engineering (and hindering or favoring it through the law), depend on the relative costs of the incumbent and new entrants. Luckily, technologists seems to agree[116] about the fact that not only reverse engineering software is very costly, but that it could likely cost as much (or slightly more) than writing the original copy from scratch.

> Indeed, the reverse engineering process, including disassembly, is so time consuming that it could take as long to reverse engineer a program as it initially took to write it, if not longer. Moreover, reverse engineering provides a programmer with only a fragile understanding of the program's operation.[117]

A typical description of the cost (in terms of time) needed to perform reverse engineering is also provided by Johnson-Laird:

> A modern program may consist of at least 300,000 […] instructions. Assuming the engineers would take only 30 seconds to decode an instruction, this means that [engineers performing decompilation] would take 2,500 hours to complete the disassembly, and that would only tell them what the raw instructions were. They would still have no high level understanding of the code itself. Nevertheless, ten months (2,500 hours) later they would have a disassembled listing.[118]

The previous description may be slightly outdated, however, at present, decompilation is still a labor intensive, slow effort. Moreover, even assuming that more phases of software reverse engineering could be automatically performed in the future, the reason for which reverse engineering is costly seems to be more logical than technological. As observed, amongst others, by Spoor,[119] it does not seem likely that the cost of decompilation will significantly drop in the foreseeable years:

> Given the tremendous demand for software maintenance, much energy has been devoted to attempts to automate the disassembling and reverse engineering process. These attempts have been far from successful, and most software specialists doubt whether the automation of these processes will ever become feasible. According to Lietz,[120] disassembly and decompilation lead to programs which may be

---

[114] In principle, the fact of having the possibility of analyzing and comparing more than one functionally equivalent piece of software could provide some useful insights to the other-potential-entrants, but we may assume – at first approximation – that the previous analysis would apply to any of them (i.e. any late-late-comer would face the same kind of trade-off as the first one).

[115] At most, they may form better expectations concerning the cost and chances of success of reverse engineering projects and they may take this into account in their possible licensing choices.

[116] Compare, among the most quoted, A. JOHNSON-LAIRD, *Software Reverse Engineering in the Real World*, 19 University of Dayton Law Review, 843 (1994), BAND & KATOH, *Interfaces on Trial*, , JOHN ABBOT, *Reverse Engineering of Software: Copyright and Interoperability*, 14 J.L. & Inf. Sci., 7 (2003).

[117] BAND & KATOH, *Interfaces on Trial*, 15.

[118] JOHNSON-LAIRD, *Software Reverse Engineering*, 873.

[119] SPOOR, *Copyright Protection and Reverse Engineering*, 1078.

[120] Indirect quotation of B. Lietz, *Technische Aspekte des Reverse Engineering: Motivation, Hilfsmittel, Vorgehensweise, Nachweisbarkeit*, 7 Computer und Recht 564 (1991).

> read, not yet to programs which can be understood. It seems highly unlikely that anything of this kind will become possible within the next few years, if ever.

Not only specialized law & economics and law & technology literature seems to agree about the cost of reverse engineering. Similar conclusions seem to have been reached by the EPO in 1993, when the Office had to decide whether the disclosure to the public of a device embedding a certain chip, describing a control procedure, also implied a public disclosure of the procedure itself. The Board concluded that:

> In theory, it is possible to reconstruct the contents of a program stored on a microchip, for example by using a 'disassembler' program or by so-called reverse-engineering. However, these procedures require an expenditure of effort on a scale which can only be reckoned in man-years […]. [T]he usefulness of the knowledge to be gained by investigating the microchip would therefore have been entirely disproportionate to the economic damage caused by the time spent on such an investigation.[121]

Moreover, it is worth noting that developers having realized a program thanks to reverse engineering may have problems maintaining it, if they did not perfectly understand all the choices of original authors. In fact, when a program has to be updated, modified or otherwise maintained, it is crucial to have access to the original comments of developers that designed the piece of software. And – as I explained in the first paper of this dissertation – these comments are completely wiped away by the compilation of source code into object code. In fact:

> The essence of these comments is that they contain higher level information included specifically to help understand what was going through the programmer's mind as he or she wrote the code. Bearing in mind that it costs approximately ten times more to maintain a program during its useful life than it cost to develop in the first place, it is easy to understand why a maintenance programmer needs all the help he or she can get.[122]

And that, again, implies that:

> A software thief who lives by reverse engineering will die a death in the marketplace because of reverse engineering. The costs of reverse engineering, taken across the product's entire life, usually five to seven years, will invariably be higher than software written de novo.[123]

To summarize, software reverse engineering could not be used to easily appropriate someone else's competitive advantages at the end of the '80s (when the Software Directive was drafted and discussed) and surely could not be used to do the same during the '90s (when several of the technologists quoted above addressed this problem). The situation does not seem to have changed today. As Spoor[124] puts it "the mere thought that at some time in the future automated software reverse engineering might become a reality, thus enabling competitors to disassemble programs to find an easy way around existing protection has been enough to scare the wits out of software owners". That is perfectly understandable; what is less understandable and certainly not excusable is that these fears, having been empty for more than 20 years and likely having negligible empirical support today, may still worry legislators and commentators all around the world. I will discuss this further in the concluding part of the paper.

Finally, it is appropriate to recall that, even once one has obtained a good approximation of some else's source code, the reimplementation of this code is far from obvious or cheap. In fact, the success of several open source projects is actually proof that successfully re-implementing a piece of software – even if source code is already available (and with a much better quality than any decompilation project could yield) – is a far from banal task. Coherently with that, the European Commission, in its Microsoft Decision,[125] noticed that re-implementing in a non-Microsoft operating system the protocols of which Microsoft was forced to disclose the specification would require significant effort:

> The example of Cisco's attempt to port Active Directory to UNIX shows that the amount of time and investment that is involved in the implementation of even a detailed and accurate specification for the protocols […] would be significant. In order to write, optimise and debug their implementations of the

---

[121] Decision of the Technical Board of Appeal of the European Patent Office (EPO) at 308. Also quoted by SPOOR, *Copyright Protection and Reverse Engineering,* , footnote 49.

[122] JOHNSON-LAIRD, *Software Reverse Engineering*, pp. 857—858.

[123] Id., p. 901.

[124] Jaap H. Spoor, *Copyright Protection and Reverse Engineering of Software: Implementation and Effects of the EC Directive*, 19 U. Dayton L. Rev. 1063, p. 1079.

[125] C(2004)900 of April 21, 2004.

> specifications disclosed by Microsoft, Microsoft's competitors would in fact have to make efforts comparable to those made by Microsoft itself as regards its own implementation work. In reality, due to the fact that Microsoft controls these specifications, Microsoft's competitors would unavoidably be at a disadvantage as regards the quality of their implementation of the disclosed specifications, compared to Microsoft's own product. There would also be an unavoidable time lag between Microsoft and its competitors, since Microsoft would only have to release the specifications when it already had a working implementation."[126]

3.2.1. If the cost of reverse engineering dropped

In some industries, trade secret may not be a good tool to protect innovation, because the cost of late comers to reverse engineer a given product may be very low with respect to the one borne by the initial developer. Indeed,

> arbitrary and irrational results flow […] from the fact that secrecy (or the lack of it) does not reliably indicate social value; they are compounded whenever a second comer's costs of reverse engineering unpatented innovation of real social value tends to approach zero for one reason or another.[127]

Fortunately, in the field of software the costs of the incumbent and of late comers are not completely unrelated. On the contrary, a product that required a significant amount of research investment is likely to be quite complex to reverse engineer and – what is probably more relevant – development (i.e. reimplementation) costs will be more or less the same for the original developer and for late comers. Hence, I argue that – as long as literal or quasi-literal copying is forbidden by copyright (following the traditional doctrines developed to deal with literary works) – we are indeed in a situation in which "trade secret law successfully mediates between innovators and borrowers of traditional industrial creations", so that "there is no perceived shortage of investment in incremental innovation."[128] That having been said, it could be theoretically possible to find that the cost of reverse engineering is so low that complete free riding on ideas and principles hidden in object code, possibly including interface specifications, would not be a first best solution, because this situation would create an excessive subsidization of late comers by original innovators, reducing the incentives to create. Alternatively (or cumulatively), we could take into account economic arguments (as a two-sided approach and the ICE paradigm, I mentioned in the General Introduction and in the first paper) to reach similar conclusions.

Several scholars discussed how to deal with the risk of excessively easy appropriation of ideas in the field of software, biotechnologies and similar high-tech domains, typically going under the label of the "new economy". They frequently refer to what Reichman[129] argued in 1994:

> The reverse engineering of industrial products by proper means was typically difficult in the nineteenth and early twentieth centuries, but new technological advances and products bearing know-how on their face are exposed to rapid duplication by competitors who expend only trivial efforts.[130]

Reichman's intuition is surely powerful. However, as shown above, the complexity of software systems apparently grew as fast (or maybe faster) than reverse engineering techniques. Moreover, even if there were perfect and automatic "decompilers"[131] recreating the original source code (something that is – at the moment – still in the field of science-fiction), development costs would still represent a powerful barrier to steal the original innovator' market.[132] Overall, I submit that software reverse engineering is still very far from being

---

[126] Id., §§ 719—721. The Commissions based its statements on Microsoft's description of the porting of Active Directory to UNIX: "As noted previously, Windows 2000 and UNIX are quite different operating systems, and it proved more difficult than Cisco expected to port Active Directory to UNIX. As a result, Cisco enlisted the aid of Mainsoft Corp. (Mainsoft), which has substantial experience porting Windows services to UNIX […]. Microsoft entered into a separate Windows 2000 source code license with Mainsoft to enable Mainsoft to assist Cisco with its efforts to port Active Directory to UNIX. Thus far, those efforts have not been successful, and it is not clear whether they ever will be. This is yet another indication of the difficulty of porting Windows 2000 services to non-Windows operating systems."

[127] REICHMAN, *Legal Hybrids*, , p. 2510.

[128] Id., p. 2520.

[129] Id..

[130] Id..

[131] Meaning, in this context, pieces of software performing automated decompilation.

[132] Otherwise, as mentioned above, it would be trivial to imitate all the technical qualities of open source platforms and nobody would care about the viral effect of the GPL and similar licenses, because it would be possible to appropriate the sources of value of open source software, without respecting the limits imposed by licenses. And this is clearly not the case.

capable of granting easy access to APIs, communication protocols, algorithms and several kinds of ideas embodied in compiled binary code.

In any case, and for the sake of discussion, let me also discuss the hypothesis in which decompilation techniques become perfect and cheap. Also in this case, I submit that we should still allow competitors to use API specifications, paying (at most) some kind of compensation under liability or quasi-liability rules.[133] I will not discuss this case much further, since – apart from being at odds with the empirical evidence and the opinion of technologists – it has already been analyzed by Prof. Reichman in 1994.[134] As I already hinted, the author discussed cases in which innovation was "near the surface" of innovative products (and hence easily appropriable), as it would be in the case of software in a world were reverse engineering is perfect and cheap. Even in this case, the author showed that the establishment of a property right is just one of the possible solutions, and not necessarily the best one:

> In reality, no compelling logical nexus exists between a chronic shortage of natural lead time and the grant of exclusive property rights. The standard legislative response will not withstand legal and economic analysis. This [paper] demonstrates that legislators can obtain all the advantages and few of the disadvantages of exclusive property rights, at far more acceptable social cost, by instituting a modified liability regime that deals directly with applications of scientific and technical know-how to industry.[135]

Hence, even if new empirical results showed that I am wrong and that significant market failures could arise from an excessively easy reverse engineering and an excessively weak copyright, an appropriate solution should still be based on Reichman's reasoning:

> the solution lies in curing the market failure in question by means of compensatory legal devices that restore and enhance competition as it would have evolved in the presence of an effective trade secret law.

---

[133] This solution can be achieved telling that there are no injunctions available (APIs are not protected by an intellectual *property* right), but that – in cases of excessive advantage by parasitism, which (I agree with Weiser about this point) may be theoretically possible in particular when interoperability is achieved by horizontal competitors and not by vertical producers of complements (even if these complements are potential future competitors[133]) – the usual rules governing unfair competition could be activated to grant some compensation to the platform owner. In this case there is an additional problem: it would be appropriate – at least – to allow for a *una tantum* compensation (instead of royalties), in order to make it possible to release the source code of compatible programs under open source licenses.

In fact, for the open source movement, an important condition for being able to re-implement interoperability information is to be able to pay it (in terms of fee to access trade secrets and/or in terms of patent fees) on a *una tantum* basis. The point is that the open source model requires free circulation and duplication of copies, hence, not per-copy fee is compatible with it. See, for instance, the Free Software Foundation Europe, press mailing list, *EU antitrust case over: Samba receives interoperability information*, December 20, 2007 http://mailman.fsfeurope.org/pipermail/press-release/2007q4/000191.html (last visited July 21, 2008): "The Samba Team has decided to make use of Micrsoft's obligation under the European judgements. Through the Protocol Freedom Information Foundation (PFIF), network interoperability information has been requested and a one-time access fee of 10.000 EUR is being paid to give Samba team full access to important specifications." See also Samba Team press release, 20th December 2007. Available at http://samba.org/samba/PFIF/ (last visited July 21, 2008): "After paying Microsoft a one-time sum of 10,000 Euros, the PFIF will make available to the Samba Team under non-disclosure terms the documentation needed for implementation of all of the workgroup server protocols covered by the EU decision. Although the documentation itself will be held in confidence by the PFIF and Samba Team engineers, the agreement allows the publication of the source code of the implementation of these protocols without any further restrictions. This is fully compatible with versions two and three of the GNU General Public License (GPL). Samba is published under the GNU GPL which is the most widely used of all Free Software licenses. In addition it allows discussion of the protocol information amongst implementers which will aid technical cooperation between engineers. No per-copy royalties are required from the PFIF, Samba developers, third party vendors or users and no acknowledgement of any patent infringement by Free Software implementations is expressed or implied in the agreement. The patent list provides us with a bounded set of work needed to ensure non-infringement of Samba and other Free Software projects that implement the protocols documented by Microsoft under this agreement. Any patents outside this list cannot be asserted by Microsoft against any implementation developed using the supplied documentation. Unlike the highly dubious patent covenants recently announced by some companies [ref. to the agreement between Microsoft and Novel] this warranty extends to all third parties."

[134] REICHMAN, *Legal Hybrids*. See, in particular, p. 2445: "To sustain optimum investment in the production of high technological goods and services without creating market-distorting legal monopolies, reformers should elaborate an improved set of ancillary liability rules. These rules should emulate the functions of classical trade secret law while rationalizing and adapting its modalities to current conditions. Besides affording artificial lead time to incremental innovators, a rationalized liability regime would allow second comers to use the formers' unpatented innovation to develop socially desirable innovation of their own, on the condition that they contributed, directly or indirectly, to the innovators' research and development costs. This new intellectual property paradigm should provide a limited, non-exclusionary form of relief for innovators who routinely apply unpatented, non-copyrightable know-how to publicly distributed industrial products. It would thereby bridge the gap between the mature patent and copyright paradigms without creating new barriers to entry or other anticompetitive effects characteristic of regimes implementing exclusive intellectual property rights."

[135] Id., 2505.

> In other words, if classical trade secret law cannot satisfactorily cope with incremental innovation bearing know-how on its face, the simplest remedy may lie in convincing the relevant community of innovators and borrowers to accept a substitute regime that provides some functional equivalent of what trade secret law provides under optimum conditions.[136]

Now, let me recall that the traditional rule, according to which there is an obligation to reverse engineer undisclosed know-how only by "proper means"[137] is not only a morally appropriate rule and a rule determined for reasons of public order. It is also a way to avoid excessively cheap appropriation (and the waste of resources that would be created by firms having to fight "improper" appropriation of trade secrets without the help of the law. If one estimates that today reverse engineering is, in fact, too cheap, legal obstacles to decompilation may be read as tool applying the solution proposed by Prof. Reichman.

Notice that, even reading existing obstacles to decompilation as tools to avoid excessively easy free riding, there is still some room for criticism. In fact, adjusting the cost of reverse engineering is a legitimate policy lever, but – in whichever direction one decides to use it – there are good reasons for using it only for a limited time period. For instance, non-competition clauses binding employees with previous employers are always temporarily limited. And the same should be true for rules increasing the cost of reverse engineering: if possible, this cost should be increased for a period sufficient to recoup sunk costs (investments, including risks), but not forever. To the opposite, obstacles to decompilation existing – for instance – in the European Software Directive are virtually perpetual. (In fact, limits to decompilation formally last as long as copyright, but the copyright term is so unreasonably long to be *de facto* unlimited, especially with respect to such a dynamic field as the software industry.)

To conclude, and coming back to the real world where reverse engineering is (for the moment) very costly and time-consuming, the only actual cases in which I could suggest to take into account something like a "liability rule" to use APIs are the ones in which a competition authority (like the EU Commission, the US DoJ or the FTC) mandates disclosure. In these cases, the natural barrier offered by trade secret is lowered in a significant way (*de facto*, artificially dropping to zero the cost of reverse engineering APIs), hence it may be meaningful to allow for compensation of the incumbent.[138]

3.3. Why competitors reverse engineer at all

At this point, one may legitimately ask why, if reverse engineering is so costly and does not offer the possibility of competing against incumbents with a significant advantage, do some people do perform it at all. This question is more than reasonable, but so are the answers. In fact, achieving interoperability (even if it is not perfect) makes it possible to access a kind of rent. That rent is created by the technology-based economic phenomena that I described in the General Introduction, with particular reference to network effects (direct and – especially – indirect). The reason why I label as "rents" the profit opportunities generated by the existence of network effects (and related switching costs) is that the investments, which created these opportunities, have typically been supported by users and producers of complementary goods and not by the developer of the incumbent software.[139] In other words, decompilation makes sense (for late comers) not as a technique to reduce software development costs, but as a tool to overcome barriers to entry created by network effects and switching costs. Indeed, if these rents were not at stake, it would be cheaper to generate a similar – but incompatible – program starting from scratch.

Hence, it is only when an incumbent reaches a position of leadership in the market and the market starts rewarding it with this kind of rent that reverse engineering becomes economically feasible. And the kind of profit that is "taken away" by reverse engineering projects aiming at horizontal interoperability is typically only in part associated with the R&D costs sustained by the incumbent; most of it comes from the "rent" generated by the investments (and sunk costs) of developers of complementary products and final users.

---

[136] Id., p. 2533.

[137] Id., text accompanying footnotes 469–487.

[138] In any case, this issue is more related to competition policy than to intellectual property: I will provide some additional elements to discuss it in the third paper of this dissertation. Consider also that – if there is no or very limited compensation for the mandatory disclosure of trade secrets – some competitors may save reverse engineering cost and achieve interoperability, free riding not only the platform owner's investments in designing APIs and CPs (which may be negligible, as the EU Commission found out in the case of Microsoft), but on the platform's investments to create a working two-sided market, for instance investing in support, assistance and documentation for third parties producing complementary products to the platform itself. About that, see § *9.3. A possible economic criticism* in the first paper.

[139] Notice, however, that the incumbent software house may have invested resources in order to encourage the creation of these network effects.

Actually, as I tried to show, research costs (in the form of reverse engineering) and development costs (in the form of rewriting the program, avoiding copyright violation and the suspect of it) of late comers wanting to achieve horizontal interoperability are likely to be higher than the original R&D costs of the incumbent. Therefore, the kind of competition that "decompilers" are likely to be able to exercise in the market is unable to menace the economic survival of the incumbent. Quite the contrary, incumbents are already in the market and more of their costs (from R&D to marketing costs) are already sunk, not to mention the fact that established incumbents may have better access to capital and credit in general. Consequently, it is the incumbent that could credibly threaten late comers with harsh price competition. This means that late comers wanting to survive in the market have either to differentiate (for instance, offering products which are more appealing to some kind of customers, like the informatics savvy people liking Linux or stylish customers liking Mac) or to innovate significantly (in terms of performances or look and feel of the programs or otherwise). Pure price competition is not likely to work, since the incumbent too may credibly drop prices to zero, given the almost negligible marginal cost of producing copies of existing software. In this sense, the entry of new players may reduce supra normal profits of incumbents, because it forces them to share the benefits of investments (in creating and buying interoperable programs) made by producers of complementary products and by customers. However, the basic remuneration of the incumbent's investments should still be possible.

3.4. Considering risk

So far, I assumed that the level of uncertainty about the market prospects for products embedding a given innovation is similar for both the first comer and any given late comer. Previous results would be even stronger (in the sense that they would imply that reverse engineering does not pose any risk of free riding) should late comers have worse market prospects than incumbents. However, serious problems may arise if *ex ante* (sunk) costs are high and there is a high degree of uncertainty for the first comers, while uncertainty is dramatically reduced for late comers (sometimes labeled as "copy-cats" or, as I prefer, functional cloners).[140] In fact, risk should be taken into account carefully: directly comparing the sunk costs of (successful) incumbents and of (whichever of the) late comers would be a mistake, if they faced significantly different amounts of risk; to perform meaningful comparisons, we should take into account also the probability of success of these different investments. For instance, if only one out of ten of the pioneers is successful, while all the late comers – investing a similar amount of money – are reasonably sure to have a marketable product by following a successful path, then the comparison becomes misleading.

As a first general remark, consider that this problem is not inexistent in traditional markets: start-up firms frequently bankrupt or close down even in the traditional footwear market, mentioned in the General Introduction, and those who experiment new marketing models are probably more likely to fail (but also to have supra-normal profits) with respect to entrants, which follow tried and tested strategies. Moreover, in the software field this problem is further reduced by the fact that uncertainty is usually not as high as in basic research or in some of the fields (like chemistry, drug production, etc.) where there is unambiguous empirical evidence of the need of strong IP protection in the form of patents.[141] In fact, when you develop a piece of software, you may have a reasonable degree of certainty about the fact that you will produce something working, even if you may not be sure of its performances or of the market response. Indeed, this is one of the reasons why patents are not the primary legal tool of protection for software innovation. Furthermore, the initial investment needed to develop a piece of software is frequently not as prohibitive as one may think, as demonstrated by the competitiveness of open source projects (at least, these projects demonstrate that barriers to entry are mainly related to programming skills and the possibility of paying a skilled workforce, not to huge capital availability). Finally, and probably more importantly, the likelihood of commercial success for a piece of software is not likely to increase that much from the fact of being a late comer. In fact, even though second entrants receive several insights from the experience of incumbents, they rarely enjoy perfect compatibility despite any decompilation effort (also various versions of the same software are frequently less than perfectly interoperable). What is more, they are likely to be at a disadvantage on reputational grounds and do not have anything especially "new" and convincing to "sell" to attract venture capital. In a few words, it is far from sure that the likelihood of success of late comers is higher than the one of the first comer. In

[140] See WENDY J. GORDON, *Assertive Modesty: An Economics of Intangibles*, 94 Colum. L. Rev., 2579--2593 (1994), p. 2586.

[141] There are several studies showing that patents are surely working as incentives (at least as surely as one can be using econometrics) in some markets, like pharmaceuticals/chemicals, while this may be disputed in several others.

this setting, also late comers will enter if and only if they can expect to gain – in case of success – much more than their development costs.

All that having been said, I admit that the analysis offered in this paper would need a serious revision, if empirical analysis showed, against my expectations, that the success rate of decompilation projects is significantly higher than the success rate of software projects starting from scratch. Fortunately, I do not see any hint in this sense, in the (admittedly anecdotal) empirical evidence that I collected and in the case studies I presented.

3.5. The (still simple) economics of decompilation after open source

Let me now consider the case in which one of the late comers follows an open source model of software development. Assume, in particular, that the first late-comer is an open source community, the goal if which is to achieve horizontally interoperability with the original system.[142] If this is the case, the open source community supporting the project would face the usual trade-off, but the "expected revenues" of the new project would be evaluated in a peculiar way. A complete analysis of the incentive structure of open source developers is outside the scope of this paper, but some scholars[143] isolated several possible explanations, including signaling incentives. More generally, the motives of open source developers are various and include:

> own use benefits, complementarity with proprietary products sold in the market, signaling, education, and social psychological motives such as altruism or simple enjoyment.[144]

In the context of the paper at hand, the relevant consequences of this different structure of incentives are that: (1) entry could occur in cases in which commercial firms would not enter (in fact, "difficult or almost desperate projects" or projects going "against a powerful incumbent" could be seen as a bad commercial enterprise by commercial software houses, but as wonderful occasions to show their talent from the point of view of open developers); moreover, (2) an open source project will publish its entire source code.

Notice that point (1) does not necessarily imply that "open source entry" would occur in many cases in which commercial firms would not dare to enter. In fact, one should notice that, normally, the incentives to produce open source software are actually lower (not higher) than the incentives to produce a commercial equivalent, since only some indirect revenues may be collected and the software is typically given away for free. Moreover, successful open source projects are frequently backed – or even directly subsidized – by commercial firms, so that their incentives may even be more similar to the ones of traditional software houses than one could expect. Finally, nothing prevents commercial firms from collecting revenues from the same sources (customization, training, etc.) from which open source developers derive their profit. Hence, there should not be excessive entry from FLOSS entrants. In any case – even if open source entry was more likely than commercial entry – no market failures would result, because of the different incentive structure of open source projects: if this structure of motivations fosters more innovation, commercial software houses should adopt it as well, and nothing prevents them from doing so (actually, they do with increasing frequency). (That having been said, I suggest downplaying the relevance of this point: there is no empirical evidence of excessively strong entry from open source firms.) To conclude about point (1), the fact that the first entrant may be an open source project should not be directly worrying in terms of reduced incentives to innovate.

It is point (2) that may be more relevant. It means that later-potential-entrants (i.e. new entrants from the second on) would have a much reduced "cost of reverse engineering", because they could free-ride on the publicly disclosed results achieved by the first open source entrant. In principle, if the open source project achieved a perfect result, the cost of reverse engineering would drop to zero or almost for other-potential-entrants. But, please, notice that this is just a potential polar case: usually, reverse engineering would not be perfect. And, even if it was, the open source project would disclose a new implementation (although in a human-readable format) with comments and not a specification document. Moreover, notice that a re-implementation (from scratch, in terms of actual written code) should be performed in any case.

---

[142] Notice that this is a polar case: cases in which just vertical interoperability is achieved would be qualitatively similar, but would have much more limited effects; also cases in which the open source functional cloner is not the first of the new entrants would be less problematic than this polar case, since the incumbent would have more time to recoup its investments.

[143] See, in particular, JOSH LERNER & JEAN TIROLE, *Some Simple Economics of Open Source*, 50 The Journal of Industrial Economics, 197--234 (2002).

[144] STEPHEN M. MAURER & SUZANNE SCOTCHMER, *Open Source Software: The New Intellectual Property Paradigm*, NBER Working Paper No. 12148 (March, 2006), p. 4.

To summarize, the worst possible case (from the point of view of the first entrant and of the incentives to innovate) is one in which a perfectly horizontally interoperable project releases its source code, including a full-fledged set of comments providing information that is equivalent to the one an interface specification document would offer. Indeed, after such an open-source-entry, other late comers would see their reverse engineering cost drop to virtually zero (but would still have to bear the cost of re-implementation). Would this create a market failure? Surely, this would make a market failure more likely than in the standard case, however, it would still be far from certain.

A first argument concerns the – already mentioned – fact that the publication of source code, when existing, does not seem to generate the impossibility of staying in the market. In fact, open source projects may resist – sometimes even as market leaders[145] – despite that fact that other firms may easily appropriate their ideas and understand the principles behind their pieces of software. Indeed, developing well-written software is both costly and difficult and it requires time. Moreover, no market failure would occur as long as:

(Cost of research) <= (gains from lead-time)

In fact, both the described case studies and the law & technology or law & economics literature suggest that several years may be needed in order to achieve a decent level of horizontal interoperability. The study of the most effective of previously mentioned projects, Wine (which is still unable to run in a usable way about 50% of Windows' applications[146]), suggests that a couple of years are needed even to implement ordinary incremental innovations, while at least 3-5 years seems to be a reasonable (educated) guess regarding the lead-time of first comers, when major technological shifts are involved.

A lead time of a couple of years, extending to 3-5 for more significant innovations, may be more than sufficient to recoup the majority of investments in the field of software. At least, that is coherent with the suggestion of the "Manifesto concerning the legal protection of computer programs", written by two leading lawyer-economists and two technologists in 1994:[147]

> Markets have a kind of 'basal metabolic rate' that determines the length of product development cycles. For software, it is currently the one or two years required to create and to test a totally new product, or the roughly twelve to eighteen month interval required to develop and to test the new release of a preexisting product.

Notice that, the *sui generis* protection that these authors proposed for software (consisting in a kind of anti-cloning provision, limited in time, but balanced by disclosure obbligations) had a suggested duration of about three (or less) years.[148] After this period, the proposed legal regime would guarantee a much broader access to the original source code than the one offered by reverse engineering. Moreover, also the possibility of reusing the original code, after the initial "artificial lead-time" offered by the sui generis protection was much more significant than the one offered by copyright (the proposed system was more similar to the special regime governing the reproduction of masks and topographies of semiconductors).

We may compare the *sui generis* protection recommended by the "Manifesto" with the protection technologically offered by secret, reinforced by the legal protection against literal copying of the implementation offered by copyright. Indeed, the latter is slightly longer and, what is more important, much more pervasive. That is true even in the most worrying case for incumbents, that is when the first late-comer

---

[145] Such an example is the Apache web server: see http://en.wikipedia.org/wiki/Apache_HTTP_Server.

[146] This rough estimate is suggested by Wine's developer Jeremy White (see WWN Interview Series: Jeremy White; available at http://www.winehq.org/?issue=347; last visited August 5, 2008) and it is based upon the users' generated database of wine applications (http://appdb.winehq.org/). "If I look back on my time with Wine (about 10 years now), this is the most exciting Wine has ever been. When I first started, the only thing you could *always* run was Solitaire. Other things could be made to work, but it took a lot of effort. Contrast that with today when, with few exceptions, it's reasonable to hope that *any* application will work. And, in practice, a substantial fraction of applications do install and run. (I don't know what that number is; I don't have a good way to quantify it). The WineHQ ratings page suggests it's close to 75% run at least as bronze: http://appdb.winehq.org/browse_by_rating.php (I think that page over inflates things, but I think that 50% is not a bad rough estimate for applications that install and at least start to work)." (Ibid.)

[147] *See* SAMUELSON, et al., *A Manifesto*, p.2408 in particular.

[148] *See* Id., p.2423 in particular. "We note that a number of countries have recently adopted or proposed short terms of protection (generally three years or less) for unregistered designs of industrial products. A concept of this sort might usefully be adopted for protecting program compilations, at least insofar as there is some minimal creativity in the compilation (that is, it does not consist entirely of standard or commonplace elements arranged in a standard or commonplace way). Anti-cloning protection of the sort we recommend for programs could be implemented by legislation or by common law."

achieving interoperability follows an open source model of software development (and hence facilitates the work of later entrants through the disclosure of its own source code).

4. Decompilation in the EU

The European Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs[149] (hereinafter, the "Software Directive") explicitly addresses various forms of software reverse engineering.

The so-called "black box reverse engineering" (i.e. the study of the working software to understand ideas, principles and methods on which it is based) is discussed at article 5, in particular at paragraph 3:

> Article 5 – Exceptions to the restricted acts
> 1. In the absence of specific contractual provisions, the acts referred to in Article 4 (a) and (b)[150] shall not require authorization by the rightholder where they are necessary for the use of the computer program by the lawful acquirer in accordance with its intended purpose, including for error correction.
> 2. The making of a back-up copy by a person having a right to use the computer program may not be prevented by contract insofar as it is necessary for that use.
> 3. The person having a right to use a copy of a computer program shall be entitled, without the authorization of the rightholder, to observe, study or test the functioning of the program in order to determine the ideas and principles which underlie any element of the program if he does so while performing any of the acts of loading, displaying, running, transmitting or storing the program which he is entitled to do.

It is undisputed that article 5.3 allows legitimate users to study the working of the program "from outside", i.e. to perform black box reverse engineering. This provision is broad, and essentially allows learning in any way from a piece of software, while legitimately using it. Moreover, this possibility cannot be excluded by contract, by virtue of article 9 of the Directive.[151] Even if the European legislators felt the need of explicitly allowing black box reverse engineering, frankly I do not see how it could have been otherwise, without creating a monopoly on mere ideas. A much more disputed issue is whether article 5.1 allows the legitimate owner to perform decompilation for the purpose of error correction. The solution reached by the majority of scholars[152] is negative, especially because decompilation is considered to be a special case, completely dealt with in article 6. That said, I will discuss this issue further below, because I am skeptical about this negative conclusion. Notice that, in any case, the decompilation exception of article 5.1 – if it exists at all – would be purpose-bound to error correction.

For the purposes of the paper at hand, the most interesting provision is article 6 of the Directive:

> Article 6 – Decompilation
> 1. The authorization of the rightholder shall not be required where reproduction of the code and translation of its form within the meaning of Article 4 (a) and (b)[153] are indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs, provided that the following conditions are met:
> (a) these acts are performed by the licensee or by another person having a right to use a copy of a program, or on their behalf by a person authorized to do so;
> (b) the information necessary to achieve interoperability has not previously been readily available to the persons referred to in subparagraph (a); and
> (c) these acts are confined to the parts of the original program which are necessary to achieve interoperability.
> 2. The provisions of paragraph 1 shall not permit the information obtained through its application:
> (a) to be used for goals other than to achieve the interoperability of the independently created computer program;

[149] A consolidated text of the Directive is available from here: http://ec.europa.eu/internal_market/copyright/docs/docs/1991-250_en.pdf (last visited August 5, 2008).

[150] These acts include: "(a) the permanent or temporary reproduction of a computer program by any means and in any form, in part or in whole. […] (b) the translation, adaptation, arrangement and any other alteration of a computer program and the reproduction of the results thereof […]".

[151] See below and Article 9 of the Directive ("Continued application of other legal provisions"), establishing that "Any contractual provisions contrary to Article 6 or to the exceptions provided for in Article 5 (2) and (3) shall be null and void."

[152] See, for instance, GIOVANNI GUGLIELMETTI, *Analisi e decompilazione dei programmi*, *in* La legge sul software, 152--201 (Luigi Carlo Ubertazzi ed., 1994), p. 159: "non è […] consentito compiere atti di riproduzione o di traduzione del codice in maniera separate dall'utilizzazione pratica del programma, e diretti alla sua decompilazione".

[153] See *supra note* 150.

(b) to be given to others, except when necessary for the interoperability of the independently created computer program; or
(c) to be used for the development, production or marketing of a computer program substantially similar in its expression, or for any other act which infringes copyright.
3. In accordance with the provisions of the Berne Convention for the protection of Literary and Artistic Works, the provisions of this Article may not be interpreted in such a way as to allow its application to be used in a manner which unreasonably prejudices the right holder's legitimate interests or conflicts with a normal exploitation of the computer program.

The first paragraph of Article 6 of the Software Directive[154] explicitly addresses software reverse engineering in the form of decompilation, authorizing reproductions of copyrighted programs "where reproduction of the code and translation of its form […] are indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs". The same article also provides that several conditions should be met in order to enjoy the decompilation "privilege": (1) the initial copy of the program to decompile must be legally owned by the reverse engineer; (2) the information necessary to achieve interoperability must not be already "readily available";[155] and (3) reverse engineering is "confined to the parts of the original program which are necessary to achieve interoperability".[156] Article 6 of the Software Directive is frequently described as a pro-decompilation measure. And the Software Directive in general as a legal text, which is especially explicit in permitting software reverse engineering. In other words, the Directive is depicted as a legal text allowing for an exception to the general principles of copyright law – that would ban decompilation – and favoring access to interoperability information. In fact, in a comparative perspective, it should be noted that in the United States fair use and other general principles of copyright allowed for decompilation and with a potentially much broader scope (i.e. to achieve much more legitimate goals) than in the European Union, where decompilation activity is bounded to the purpose of achieving interoperability. The purpose-bound nature of the exception spelled out in article 6 is actually contrary to several basic principles of copyright law and trade secret law, in particular to the idea/expression dichotomy and to the general favor for reverse engineering, as a tool to discourage the use of trade secret and encourage other forms of legal protection requiring a full disclosure of achieved innovations.[157] Indeed, according to some authors, decompilation should also have been allowed in civil law countries, on the basis of reasoned application of general principles of copyright law.[158] But the majority of theoretical discussions in this domain

[154] Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs.

[155] This condition has been widely discussed in the literature, but without reaching a clear conclusion. Some authors (e.g. FABRIZIO BROCK, *La disciplina del 'reverse engineering' nella legge di attuazione della Direttiva CEE sul software*, 1 Rivista di Diritto Industriale, 267--279 (1993), p. 276) actually argued that there is an obligation for the potential decompiler to ask for this information to the copyright holder and possibly even that the information remains "readily available" if the owner asks to pay a "reasonable and equitable fee" in order to access it. Today it seems to me that the majority of the literature agrees with GUGLIELMETTI, *in* Analisi e decompilazione, and GUGLIELMETTI, *L'invenzione di software (2nd ed.)*, , in arguing that the potential decompiler is not even obliged to reveal its intention to access the interoperability information to the original developer: this information is "readily available" if and only if it can be easily accessed on the software documentation and/or from similar sources (published manuals, the developer's or distributor's website and so on); otherwise, the decompilation exception of article 6 applies.

[156] It is widely acknowledged by virtually all the commentators I am aware of that this condition does not prevent the decompilation of the entire software if this is necessary in order to individuate the interfaces (see, for instance, BROCK, *La disciplina del 'reverse engineering'*, p. 277). However, when it is technically possible to understand that a given portion of the code is not useful for interoperability purposes, the decompilation activity should stop.

[157] The so called "contract theory" about the origin of patent protection justifies the costs of the temporary monopoly created by patents precisely on the ground that it discourages innovators from keeping their inventions secret and the best ways to implement them.

[158] About Italy, see generally BROCK, *La disciplina del 'reverse engineering'*, pp. 270—272, arguing that "[i]n conclusion, the application of general copyright principles leads to excluding that decompilation could have been qualified as a violation of the exclusive rights owned by the original program rightholder". The reasoning of the author is based on the fact that no prejudice of the legitimate rights of the original copyright holder can derive from decompilation (unless the decompiled code is distributed) and on the fact that any formal violation is taking place in the context of a private use that is analogous to the translation of a foreign work done for personal use, for instance to practice a foreign language (I would actually note that translation – even if performed for private purposes – generates a derivative work and it is not completely undisputable that this could formally amount to a copyright violation). In any case – coherently with what I have argued in the first paper – the author explicitly stresses that any product applying the principles learned from decompilation should be expressed in a new and independently created form and this is why no copyright violation would take place.

Some elements that could have been used to argue in favour of the legitimacy of software reverse engineering (absent any specific law concerning copyright protection of software) could be found also in GUGLIELMETTI, *L'invenzione di software (2nd ed.)*, p.204 and in other authors arguing that – strictly speaking – the object code should not be considered as a derivative work of the source code

have been silenced by the Software Directive and by article 10 of the TRIPs Agreements, all clearly stating that computer programs shall be protected as literary works, "whether in source or object code".[159] According to the standard interpretation of these norms, intermediate copying of the object code in order to create a more understandable derivative work are copyright violations, unless some exception or limitation applies. Hence, far from limiting itself to clarifying the legitimacy of software reverse engineering, the Software Directive may actually be seen as the norm clarifying (in Europe) that software reverse engineering techniques involving intermediate copying of expression (i.e. decompilation) violate – in general – copyright law. Only having established that, the Directive introduced a precise, purpose-bound and quite narrow, exception to allow some instances of decompilation.

The aforementioned conditions spelled out in the first paragraph of article 6, apart from being rigidly purpose-bound (i.e. limited to the goal of achieving interoperability), are similar to the ones required by US courts to find a fair use exception for reverse engineering.[160] Or – at least – they would be, if they were interpreted in the same broad way in which fair use is applied in the US. In fact, if article 6 consisted only of paragraph 1, the main differences with the US norms would be that: (1) in the EU there is an express statutory provision, while the US situation is based on case law and on the general application of the fair use doctrine (already described and discussed in the first paper of this dissertation); (2) on the one hand, the absence of specific provisions in the US is increasing flexibility and allowing a more economically based analysis; on the other hand, the existence of an explicit provision in the EU increases legal certainty; (3) in the EU there is a clear provision preempting contract clause against reverse engineering,[161] while it is far from clear that these clauses are preempted by Federal IP law in the US.[162] Indeed, if article 6 consisted of only a paragraph, one could argue that software decompilation would be more frequently allowed in the EU than in the US. That would be true, since the achievement of interoperability is, in any case, the main motive for embarking on a costly decompilation effort[163] and since the explicit preemption of contractual clauses excluding the possibility of performing decompilation is a very significant advantage for reverse engineers.

What is more interesting (and complex) to analyze (and eventually criticize) about the European Software Directive are the second and third paragraphs of Article 6. However, economists, business scholars and practitioners, but also legal scholars, have frequently neglected these paragraphs, and in particular Articles 6.2.b. The most relevant implication of these paragraphs concerns an *ad hoc* regulation of the circulation of the information obtained through decompilation. Information, which is typically not protected by copyright, nor by patents, but only as a trade secret. Indeed, I will argue that article 6.2 establishes an unprecedented and unparalleled (in other fields of technology) reinforcement of trade secret against reverse engineering.

Coming back to the norm, the second paragraph of Article 6 states that:

> [t]he provisions of paragraph 1 shall not permit the information obtained through its application: (a) to be used for goals other than to achieve the interoperability of the independently created computer program; (b) to be given to others, except when necessary for the interoperability of the independently created computer program; or (c) to be used for the development, production or marketing of a computer program substantially similar in its expression, or for any other act which infringes copyright.

(in fact, the object code is deprived of all the expressive choices of the author and transformed in a purely utilitarian object, precisely describing the operations that a computer must execute).

[159] Article 10(1) TRIPs: "Computer programs, whether in source or object code, shall be protected as literary works under the Berne Convention (1971)". Similarly, Article 1(2) of the Council Directive 91/250/EEC states that "Protection in accordance with this Directive shall apply to the expression in any form of a computer program", with an implicit but clear reference to both source and object code. Similar principles had been established by US case law, starting with Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983).

Hence, the *fictio iuris* concerning copyright protection of object code as a literary work has been codified at the international level. For comments about the goals and limits of this "metaphor", *see* in particular GUGLIELMETTI, *L'invenzione di software (2nd ed.)*, pp. 222-224 and 231-243.

[160] *See* ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 1*, 22 European Intellectual Property Review, 7-16 (2000) and ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 2*, 22 European Intellectual Property Review, 56-68 (2000). For additional details about the US and EU approached to reverse engineering, *cf.* SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*, and GUGLIELMETTI, *in* Analisi e decompilazione.

[161] See *supra* note 151.

[162] See C. R. McManis, *Intellectual Property Protection and Reverse Engineering of Computer Programs in the United States and the European Community*, 8 Berkeley Technology Law Journal, 25 (1993) and § 9.1 of the first paper of this dissertation.

[163] See also § *3.3. Why competitors reverse engineer at all.*

Letter (a) clearly reinforces the purpose-bound limitation already stated in paragraph 1. Not only decompilation can be performed just when it is "indispensable to obtain the information necessary to achieve the interoperability", but the results of decompilation could be used only to achieve this goal. At first glance, this condition may appear to be plain and reasonable, but – thinking more about it – it seems to mean that ideas, methods and abstract principles learned during decompilation cannot be reused by reverse engineers. Indeed, notice that letter (a) cannot refer to the use of protected expression accessed through decompilation, since the (obvious) status of copyrighted expression of this information is already restated and clarified by letter (c) of paragraph 2 (stating that, in general, the re-implementation of any interoperability specification cannot infringe copyright; or, economically speaking, that one cannot free-ride on the sunk, up-front cost of writing code implementing the discovered ideas).[164] The protection of ideas, methods and principles offered by the combination of article 6.1 and 6.1.a is difficult to inscribe in the traditional intellectual property environment, where trade secret is discouraged and disclosure (through patent applications) encouraged and where ideas are otherwise supposed to be free. Maybe, this kind of reinforcement of trade secret in the field of software could have looked reasonable to European legislators, because – in principle – software was not patentable in Europe, so that this legal reinforcement of the simple copyright & secret couple seemed necessary. However, at present – and as briefly discussed in the first paper[165] – the European Patent Office is systematically granting patents for software implemented inventions; hence, this additional layer of protection for ideas and technical solutions embedded in software becomes even more difficult to justify.

Article 6.2.b implies various things. First of all, and quite clearly, it leads to the fact that undertakings enjoying the decompilation exception are not free to sell the "derivative works", generated from the original program through reverse engineering (i.e. the reconstructed approximation of the original source code, that is the reconstructed *implementation*). However, this is quite obvious and descends also from ordinary copyright rules (additionally recalled by letter (c) too). In fact, letter (b) also means that it is not possible to "give to others" the reconstructed interface *specifications* or other unprotected ideas, methods and principles learned during decompilation. In practice, the protection of these elements accorded by letter (a) extends to everything, apart from the information needed to achieve interoperability; moreover, it covers information necessary to achieve interoperability as well, when disclosure to third parties is involved.

Despite being a quite unusual rule in the copyright field, the fact of allowing reverse engineering but forbidding the disclosure of its results may be seen as an application of a possibility discussed by Samuelson and Scotchmer[166] in their analysis of the law & economics of reverse engineering:

> A […] policy option is to allow reverse engineering but to forbid publication or other disclosures of information obtained thereby. For the most part, the law has not had to address this issue because reverse engineers have generally had little incentive to publish or otherwise disclose information they learn from reverse engineering. Reverse engineers have typically kept the resulting know-how secret for competitive advantage.[167]

In other words, following this policy option, each new entrant must bear the entire cost of reverse engineering on its own, and this generates two effects. On the one hand, the protection of the lead-time of the incumbent is sometimes increased (and – in any case– never decreased). On the other hand, decompilation efforts may be duplicated and that "leads to more socially wasteful costs unless the software developer licenses interface information to foreclose the decompilation effort." [168]

As I discussed in describing some simple economic principles behind reverse engineering, and as confirmed by the previous quotation of Samuelson and Scotchmer, the likelihood that reverse engineers want to let later-comers free ride on the information that they costly obtained is very limited. In principle, one could argue that the EU rule may have – more or less willingly – anticipated the case of an open source new entrant, which would be forced by its licensing scheme to release the source code of the newly realized program and, hence, a large part of the information obtained through decompilation. However, this "rationalisation" of article 6.2.b is probably not tenable, since there is an exception to the prohibition of "giving to others" the information obtained through decompilation and that exception precisely applies when the information is "necessary for the interoperability of the independently created computer program". And I

---

[164] To be sure, patented inventions would not need the additional protection offered by letter (a), hence this provision may not be have been thought for patent protected solutions.

[165] See paper 1, section 9.2.

[166] See SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering.*

[167] See Id..

[168] See Id..

submit that this entails that the entire source code of a horizontally interoperable product could be disclosed. Indeed, any kind of interoperability information needed to interoperate with the original decompiled product is also needed to interoperate with the horizontally interoperable one. And, since the reverse engineers were prohibited from making use of other kinds of obtained information (i.e. unnecessary for interoperability), the source code of the horizontally interoperable product cannot disclose any of this information.[169] The only way to argue differently would require saying that horizontal interoperability is not a legitimate goal under article 6, so that only vertical interoperability would be achievable thanks to this exception.[170] I do not think that this is the case, but I will come back to this issue later.

So, what does article 6.2.b really mean? First of all, and despite some commentators opting for the opposite solution,[171] I submit that article 6.2.b implies that selling the information obtained from a decompilation project to someone else would violate article 6 of the Software Directive. That having been said, notice that collective decompilation projects are perfectly possible (obviously, as long as the other conditions spelled out in article 6 are respected by all participants and/or there is an appropriate delegation, following article 6.1.a). In practice, the discriminant is the fact that there must be an agreement in sharing the cost of a decompilation project *ex ante* (i.e. before the project starts). Of course, this may make some economic sense, because two firms acting together are equivalent to a single firm. Instead, *ex post* (i.e. once decompilation has been performed), the cost of reverse engineering would be sunk and there could be (even if I doubt that this would frequently be the case) an incentive to sell the result of the decompilation activity for an excessively low price. In particular, this interpretation implies that reverse engineers in financial distress, or deciding to abandon their project, cannot sell the information they have obtained, and this may be appropriate, since – in these cases – the acquisition of the information could be significantly cheaper than the cost of independently acquiring it through decompilation.

An interesting interpretative question concerns the effect of the possible violation of one or more of the conditions stated in article 6. The violation of conditions contained in article 6.1 will clearly result in a violation of the copyright on the original program. What is less clear, as observed by Guglielmetti,[172] is whether the same applies to the violation of conditions stated in paragraph 2, since these conditions concern activities which are performed after decompilation itself. The very fact of dividing these conditions in two different paragraphs confirms that the second paragraph deals with somehow different issues. In fact, the case discussed in letter (c) does not present special problems: it surely concerns a copyright violation, but not one performed during decompilation. Instead, it is about the possibility of violating copyright during reimplementation. However, letters (a) and (b) are more problematic, because they concerns raw information, constituting a trade secret, but which is not copyrighted. According to Guglielmetti,[173] another significant difference between the conditions stated in paragraphs 1 and 2 of article 6 is the fact that the respecting of the first ones has to be proven by the reverse engineers (because, otherwise, a finding of copyright infringement may be presumed). Instead, the original copyright holder must prove the violations of conditions stated in paragraph 2. I strongly support this interpretation, since this is coherent with the general rule, according to which ideas, principles and methods are freely appropriable, unless an unfair appropriation can be proven. Hence, it seems reasonable to conclude that – while the violation of the conditions stated in paragraph 6.1 imply the illegitimacy of decompilation and a copyright violation and the violation of condition

---

[169] Thus, the entire source code can be made public, as I will discuss specifically in § *4.1.1. Does article 6 allow the disclosure of source* code.

[170] R. J. HART, *Interoperability Information and the Microsoft Decision*, 28 European Intellectual Property Review, 361--365 (2006), among others, seems to argue in this sense.

[171] *See* BROCK, *La disciplina del 'reverse engineering'*, p. 278. The author argues that "doubts may exist about the legality of the onerous communication of the data obtained through decompilation by the developer of an autonomously developed software to another. […] About this issue, it is necessary to distinguish two hypothesis, depending if the subject interested in acquiring this information may himself enjoy the decompilation exception or not. […] [I]n the first hypothesis, the answer [to the question whether selling the obtained information is legitimate] must be affirmative: in fact, is that operation may legitimately be delegated to authorized third parties [see article 6.1.a], there is no reason to impede the acquisition of the data resulting from the decompilation performed by others." (My tentative and rough translation.) However, this position of Brock is compensated for by the fact that the author argues that before performing any decompilation activity the reverse engineers are required to ask the creator for the necessary interoperability information (see supra note 155). And also that potential decompilers should even consider "readily available" interoperability information that is available from the creator in exchange for a fair compensation ("naturally as long as the required compensation is reasonable and equitable").

[172] See GUGLIELMETTI, *in* Analisi e decompilazione, p. 179 ff. and B. CZARNOTA & R. J. HART, *Legal Protection of Computer Programs in Europe: A Guide to the EC Directive*, (Butterworths Tolley. 1991), p. 81.

[173] See GUGLIELMETTI, *in* Analisi e decompilazione, p. 180.

6.2.c implies only a copyright violation – not respecting letters (a) and (b) of article 6.2 "just" entails the violation of a trade secret.

To conclude the comment of articles 6, we come to paragraph three (6.3):

> In accordance with the provisions of the Berne Convention for the protection of Literary and Artistic Works, the provisions of this Article may not be interpreted in such a way as to allow its application to be used in a manner which unreasonably prejudices the right holder's legitimate interests or conflicts with a normal exploitation of the computer program.

This is clearly a common principle of almost any jurisdiction, but – for instance – US courts interpreted similar arguments in a very restrictive way, allowing not only "vertical interoperability" (that is, interoperability with complementary goods, like another application for a given operating system), but also "horizontal interoperability" (allowing, for instance, decompilation to be used to realize a competitive operating system, which is able to run the same applications of the first one, maybe running on a different hardware[174]). In general, the principles stated in article 6.3 could be a good basis for a "rule of reason" analysis (as the fair use test is constructed in the US), but it looks redundant (and only provides additional constraints) in the context of a very detailed exception, as the one provided by article 6. In other words, a potential infringer cannot use article 6.3 as a defense (saying that he did not actually cause any loss to the right holder with his actions), but the right holder can use it as a sword to invoke liability also in cases which would be otherwise protected by the exception (for instance, some cases involving horizontal interoperability with complete cloning of a set of APIs). If fact, one has to conclude that the true goal of article 6.3 is just to clarify the compatibility of the interoperability exception with international treaties.[175]

---

[174] I added this specification, because the US decision I am referring to (Sony v. Connectix) concerned a case in which "horizontal interoperability" allowed to play the same applications (games) on a different hardware: it is not clear, from the case, if this fact has been determinant; it surely did weight in the direction of fair use (and – to use the language of the Directive – against the finding of an "unreasonable prejudices to the right holder's legitimate interests").

[175] Paragraph 3 of article 6 is not just a residual clause. Indeed, almost any copyright-infringing use of the decompilation exception is already excluded by other norms, in particular by 6.2.c, which is the true residual clause, since it states that the information obtained through decompilation cannot "be used for the development, production or marketing of a computer program substantially similar in its expression, or for any other act which infringes copyright". Article 6.3, as some commentators observed, is a norm helping interpretation, which is addressed in particular to foreign authors from states which are member of the Berne Convention (which has also been incorporated in the TRIPs agreements of the WTO). Indeed, this norm clarifies that the decompilation exception is compatible with the Berne Convention (and that it should be interpreted in such a way that it remains compatible with it). Hence, the norm does apply to any piece of software circulating in the European Union, because foreign authors cannot claim any especially favorable treatment (their works are subject to the exception, as the ones of nationals). (For an particularly clear analysis about this point, see Guglielmetti, *in* Analisi e decompilazione, 198—201.)

Actually, some authors contest the compatibility of Article 6 with the Berne Convention. See Cornish, *Inter-operable Systems*. I anticipate that these doubts are not shared by the majority of the literature and that nowadays the compatibility of Article 6 of the Software Directive with international copyright treaties is generally accepted. However, I want to describe the reasons that generated these doubts, since replying to them shows that not only Article 6 as-it-is is compatible with the Berne convention, but also that a more general decompilation exception (that I will recommend) would be perfectly acceptable in the international setting.

To respect the Berne Convention, limitations and exceptions to exclusive rights under national copyright laws must respect the well known three-step test. This clause (included in Article 13 of TRIPs) reads: "Members shall confine limitations and exceptions to exclusive rights to certain special cases [1st step] which do not conflict with a normal exploitation of the work [2nd step] and do not unreasonably prejudice the legitimate interests of the rights holder [3rd step]." Decompilation is surely a "special case" and concerns uses of a piece of software, which are far from "normal" or ordinary. In fact, the main argument of those who criticize the decompilation exception is an economic one and concerns the third step of the test. (For a detailed discussion also of the first and second step, *see* Guglielmetti, *in* Analisi e decompilazione, pp. 199—200.) According to this argument at least decompilation to achieve horizontal interoperability should be forbidden by paragraph 3, since this kind of decompilation not only has a commercial purpose, but it also results in the creation of a product competing with the decompiled work, hence "conflicting" with the right holder's ability to exploit his/her work and recoup his/her investments. However, even if it is true that the achievement of horizontal interoperability could reduce (potentially in a severe way) the profits of the original author, the implication that this is incompatible with the Berne Convention derives from the confusion – already discussed at length in the first paper – among access to interoperability information and its reimplementation. Indeed, the "normal exploitation" of a piece of software consists in its sale/licensing to potential users. The copies and the derivative works, which are created during decompilation, do not reduce these sales, since a precondition to enjoy the exception is precisely to be a legitimate owner/licensee of the program to decompile. Clearly, it is true that decompilation may be a preliminary step to compete with the incumbent, but avoiding this kind of competition is not a rightholder's "legitimate interest". Here, I remand to the first paper for a more rigorous discussion of this point and I limit myself to providing an example clarifying my argument by analogy. Think about a visually impaired person needing a Braille version of a book in order to read it. Using a PC and some other device to obtain the Braille copy of the regularly purchased book may very likely constitute a fair use in the USA and would likely deserve a specific copyright exception in civil law countries: such an exception would certainly respect the conditions imposed by the Berne Convention. That is true, in particular, if

4.1. Vertical and horizontal interoperability

I already hinted at the fact that the "interoperability" that the Software Directive wants to facilitate encompasses both direct (or vertical) and indirect (or horizontal) interoperability. In this paragraph I discuss this issue and my interpretation in greater detail.

First of all, my reading is backed by the literal text of the Directive. Indeed, article 6 limits the rightholder's exclusive powers, when that is necessary "to achieve the interoperability of an independently created computer program with other programs", that is "with other programs" in general and not just "with the decompiled program". Notice also that the text talks about a single program (that could be decompiled) and a single rightholder (the permission of which is not necessary), hence it is not easy to argue that the plural "other programs" always refers to the decompiled ones. Additionally, this punctual literal element is confirmed in the Preamble, where recital 21 clearly talks about "reproduction of the code and translation of its form […] indispensable to obtain the necessary information to achieve the interoperability of an independently created program *with other programs* [plural]". Moreover, and despite the fact that some commentators interpreted the very same legal text, suggesting that article 6 actually allows only vertical interoperability,[176] the reading I propose is strongly backed by the general reading of the Directive, proposed by the Commission (and confirmed by the Court of First Instance) in the recent Microsoft Case.[177]

In its Microsoft Decision, the Commission explicitly referred to the concept of interoperability "as defined in the Software Directive", and then explicitly included elements of horizontal interoperability in this understanding of interoperability. For instance, at recital 277—279 the Commission criticized Microsoft's limited disclosures because "[t]hese disclosures […] have been strictly limited […] to client-to-server communication", while "the Windows domain architecture involves client-server and server-server interconnections and interactions that are closely interrelated." In other words, in order to achieve a satisfying level of interoperability, the Commission wanted Microsoft's competitors to be able to interact at any level of the software architecture, not only connecting to some Microsoft operating systems, but also being able to potentially substitute them. To be sure, in my opinion no duty to actively favor interoperability directly follows from the Software Directive,[178] but – if interoperability means what the Commission is describing in the Microsoft Decision – then there is no doubt that both vertical and horizontal interoperability are allowed under article 6 of the Software Directive.

Of course, I have to stress that the actual broadness of the concept of interoperability in the Software Directive is now widely debated. While some commentators argue that the Commission is stretching the concept in an unprecedented (an probably dangerous) way,[179] other scholars are even trying to read in the Directive a general obligation (not limited to dominant firms) to allow interoperability (not only to let other firms decompile).[180] The verdict of the Court of First Instance on the Microsoft case (delivered the 17th of

---

the visually-impaired person also bought an original copy of the book and if he acts according to several additional restrictions, including the fact that this Braille copy may not be reproduced or sold without the right-owner's permission. But what is relevant here is that the blind person of this example, having read the Braille version of the book, is not restricted in any way (whit respect to what would happen with an ordinary reader) in the uses that he/she can do of the ideas and methods learned reading the book. And this freedom is not altered by the fact that he/she needed to take advantage of a fair use or other copyright exception in order to access these ideas. Now, imagine that this person decided to write a ferocious and well-founded critique of the book, with the effect of almost annihilating its sales. Logically speaking, the critique may be a consequence of the possibility of reading the book, and this possibility has been granted through a fair use. Nevertheless, this market destructive "indirect effect" of the fair use allowing the person to read the book in the first place would never be taken into account by a judge evaluating the fairness of realizing a Braille version of the book.

[176] See, in particular, HART, *Interoperability Information*.

[177] I refer to the *Microsoft IV* case: *Case COMP/C-3/37.792 Microsoft*, that led to the *Commission Decision of 24.03.2004 relating to a proceeding under Article 82 of the EC Treaty*. The European CFI delivered its judgement on Microsoft's Appeal on the 17th of September 2007, substantially confirming the approach of the Commission.

[178] This is – at least – my opinion and the opinion of the majority of the literature of which I am aware, but see D. Caterino (footnote 180) for a broader reading of the relationship between the Software Directive and an actual duty to allow interoperability.

[179] See, for instance, Robert. J. Hart, "Interoperability Information and the Microsoft Decision", EIPR, Vol. 28, Issue 7, July 2006.

[180] See DANIELA CATERINO, *Software e rifiuto di licenza del codice sorgente*, Annali Italiani di Diritto d'Autore, 388 (2004). The author stresses that the notion of interoperability adopted by the European Legislator is very broad, but she also aliments the ambiguity that a "disclosure of source code" is necessary to have interoperability; technically this is not true: even if a full disclosure of the source code is surely sufficient to achieve interoperability, it is not necessary [but I concede that a partial disclosure of source code – or a burdensome decompilation work – may be needed, if API specifications are not up to date or if there are bugs in the original implementation]. The article is also making clear that mere right to perform reverse engineering and to implement anew a set of APIs is not sufficient to solve any problem: "*E' evidente che in siffatte condizioni la coazione al rispetto del mero obbligo di pati, contemplato dalla*

September 2007) seems to fully upheld the approach of the Commission and does so without the need to attach two different meanings to the concept of "interoperability" in the context of the Software Directive and of EU competition law. If – as it seems now reasonable to assume – the understanding of the Commission is correct, article 6 of the Software Directive undoubtedly (even if implicitly) allows competitors to reverse engineer a software program to achieve any kind of interoperability, including – for instance – to (indirectly) achieve interoperability with the software which are interoperable with the decompiled one. In other words, article 6.3 cannot be read in a way preventing horizontal interoperability. Notice that a general favor of the European legislator for interoperability, could also be recognized in the Directive 2001/29/EC of the European Parliament and of the Council of 22 May 2001 on the harmonization of certain aspects of copyright and related rights in the information society.[181]

The fact that horizontal interoperability is allowed by the Directive may also be a significant argument in deciding whether the Directive permits the disclosure of the source code of open source software realized also thanks to decompilation.

4.1.1. Does article 6 allow the disclosure of source code?

The economic sustainability of such an open model as the open source one was surely unexpected at the time of the drafting of the Software Directive. Indeed, for the majority of commentators, the success of the open source model of software development was actually more than questionable until a few years ago (or still is!). Hence, the drafter of article 6 of the Software Directive could not have reasonably taken into account the possibility that an undertaking could reverse engineer an existing platform, for instance to achieve vertical interoperability, and then "give away" for free some of the interoperability information incidentally collected during the process, even if this information was not directly useful to him. Obviously, it was easier to envisage the case in which the reverse engineers wanted to sell the obtained information, but the legislator probably thought that – as a matter of fairness – the subject entitled to receive this kind of payment was always the creator, hence the prohibition concerning disclosure of information not needed to ensure interoperability with the independently created program.

The typical scenario in a setting in which there is competition among commercial (closed) platforms is one in which the late comer could try to attract more direct and indirect network effects than the incumbent, and that could lead the late comer to release more interoperability information than the incumbent. However, there are no reasons to assume that a commercial late comer would disclose dramatically more information than the incumbent. Hence, at the time of drafting the Directive, the legislator (and probably also some of the groups lobbying in Brussels) probably thought that the rule embedded in article 6.2.b would have maintained secret a significant part of the information collected through decompilation. Indeed, given the significant investment needed to decompile existing software, who would renounce to almost the entirety of the competitive advantage coming from the fact of being the first reverse engineer? Even in the case of horizontal interoperability and decompilation of an existing software platform, the late comer would not be likely to disclose much more than the incumbent would: in fact, duopolistic competition with the incumbent looks (almost by definition) more lucrative than eliminating any barrier to entry in the market, increasing competition. However, open source projects do release their entire source code, because their commitment to openness is very strong, doing so is a precondition to get support from the community of developers and

---

direttiva (divieto di impedire od ostacolare le attività di black box analysis e reverse engineering) non è sufficiente a garantire la perfetta conoscenza delle interfacce, ovvero il corretto funzionamento del programma; pertanto, entro questi limiti concettuali deve a mio avviso ritenersi che il titolare del diritto sia destinatario di un vero e proprio obbligo di facere; cui corrisponde, sul piano antitrust, la possibilità per le autorità comunitarie di imporre al titolare del diritto non solo la rimozione di tali misure ed ostacoli, ma altresì un comportamento positivo dell'impresa dominante, un vero e proprio obbligo di disclosure." What is blurred in this article is the distinction between the obligations of a firm holding a dominant position and the general obligations of an undertaking active in the software market: the "obligation to do" – *obbligo di facere* – requested by the EU Commission to dominant undertakings seems to be disproportionably extended to any software house.

[181] Recital 54 reads: "Important progress has been made in the international standardisation of technical systems of identification of works and protected subject-matter in digital format. In an increasingly networked environment, differences between technological measures could lead to an incompatibility of systems within the Community. Compatibility and interoperability of the different systems should be encouraged. It would be highly desirable to encourage the development of global systems." For additional comments and to better contextualize this statement in the general interoperability debate, see MIKKO VÄLIMÄKI, *Software Interoperability and Intellectual Property Policy in Europe*, 3 European Review of Political Technologies, 1--11 (2005).

testers and – in any case – that is a required condition in order to (so to speak) "free ride" on the contributions of the majority of open source projects, released under copyleft licenses such as the GPL.[182]

A first question to be answered concerns the compatibility of any open source approach with the restrictions of article 6.2.b. It looks undisputable that the information needed to allow interoperability with the independently created program might be disclosed, even if this information has been originally obtained by reverse engineering another piece of software. However, does this mean that any kind of comments – potentially concerning the interoperability requirements of the decompiled software – can be disclosed? Whatever answer we provide, we risk having a paradox.

On the one hand, if publishing the source code of the interoperable software is legitimate and if – as I argued – horizontal interoperability is a legitimate goal (as the vertical one), then the restrictions of article 6.2.b concerning the disclosure of the obtained information can be easily avoided by the open source world. In fact, any open source project may claim to be working on a competing platform sharing all the APIs and the communication protocols of – say – Microsoft Windows. Having done that, the publication of any result concerning the APIs and CPs of Windows becomes licit, at least as long as a tentative implementation of these APIs is available on the new operating system (in order to couple the disclosure with an "independently created program"). Actually, since the structure of APIs and CPs is frequently disclosed to potential producers of complementary products even before their final and perfectly working implementation in the code (in order to allow them to design the architecture of their future interoperable programs), there are several arguments in favor of allowing even the disclosure of interface specifications that will be implemented, well before their actual implementation. Indeed (as observed by Giannelli)[183] there is nothing in the Directive that strictly dictates the timing of the various activities of decompilation, reimplementation and disclosure (even if all the stated conditions must be respected). Hence, if an early disclosure of specifications – with respect to full working implementation – is commonplace in a given industry, there are no reasons to impede it in case of specification information obtained through reverse engineering (because deciding otherwise would impose a significant and unjustified competitive burden on the late comer). That seems to be essentially confirmed by the comments of the Commission to the Sweden implementation of the Directive[184]:

> The Swedish implementation is only defective in that the phrase "independently created program" is missing from the transposition of Article 6 (1). It would appear, however, that this omission has a significant effect. The missing element was provided in the Directive to ensure that any decompilation of a target program does not occur before the independently created program exists (even if only in preparatory design material form).

Despite the fact that the Commission stresses the need for the existence of an "independently created program", it also seems to allow for its existence only at the stage of preparatory design material. And there is also evidence that the final design of the program could take into account the information acquired through decompilation, as it would be necessary for an operating systems aiming at being horizontally interoperable with a decompiled competitor.

On the other hand, if the reimplementation of discovered interface specifications was not allowed in an open source model, then article 6.2.b would exclude a significant part of the software world from the benefits of decompilation. In fact, open source systems are not only significant competitors of existing incumbents in terms of market shares.[185] What is more important is that open source projects are frequently the most credible threats of precisely those incumbents that look steadily established as market dominants. I will come back to this point in § *7. If uncertainty is significant only for open source projects, should we really care?*. For example, if there is a credible alternative to Microsoft Windows and Microsoft Office for IBM-compatible (or "wintel") personal computers it is only because of the existence of open source projects, like Open Office.[186]

---

[182] In fact, copyleft licenses require derivative works (and/or works embedding the licensed one) to be licensed under the same license (and to publish their source code): this is the so-called viral clause of the license.

[183] GIANVITO GIANNELLI, *in* Commentario Breve alle Leggi su Proprietà Intellettuale e Concorrenza, (Luigi Carlo Ubertazzi ed., 2007).

[184] COM/2000/0199, Report from the Commission to the Council, the European Parliament and the Economic and Social Committee on the implementation and effects of Directive 91/250/EEC on the legal protection of computer programs.

[185] Quote the market shares of Linux, Firefox, but – in particular – Apache, MySQL and PHP.

[186] OpenOffice derives from the proprietary office suite StarOffice, realized by the German company StarDivision. In 1999 Sun Microsystems purchased the source code of StarOffice and started distributing it as a freeware product. However, after about a year, Sun Microsystems announced that it was going to make the source code of its office suite available under the open source

The paradox I mentioned above derives from the fact that article 6.2.b risks being either irrelevant for open source projects (if horizontal interoperability is legitimate and the publication of source code and of any associate documentation is completely free) or unable to attain its intended results in terms of real possibilities of achieving interoperability (if horizontal interoperability is not allowed or if it cannot be achieved by open source projects). At the end of the day, article 6.2.b may be liable of generating just the paradoxical result of protecting precisely the quasi-monopolistic incumbents that sometimes dominate the less dynamic software markets.

Notice that article 6.2.b is not a problematic norm only as far as the analyzed polar cases are concerned. Intermediate "nuanced" cases are not helpful either. For instance, assume that the correct interpretation of the norm would be that publication of the source code is allowed, but only as long as the contained comments are not "too revealing" of discovered interoperability information that is not strictly needed to achieve actual and immediate interoperability with the independent program. Interpretations of this kind would generate significant uncertainty and – if coupled with aggressive communication strategies on the incumbent's part – also FUD (I will come back in the third paper on some measures to reduce strategic creation of FUD by incumbents). And the effect would either be to create some minor problems to open source projects fighting dominant incumbents (which look already more than strong enough in software markets) or to be almost irrelevant, but still capable of creating some uncertainty and potential litigations.

To summarize, it must be doubted that publishing the source code of software obtained (also) through decompilation may be incompatible with the Software Directive. In general, the publication of the Source Code will be exempted by article 6.2.b. However, some notes and comments attached to the source code may not be freely publishable, if they explain more about the decompiled software than what is needed to interoperate with the independently created open source program. Moreover, some doubts may exist concerning the possibility of disclosing interoperability information which has been obtained through reverse engineering and transformed into API and CP specifications, but which has not yet been concretely implemented in the independently realized software.

4.2. "The movement is everything, the ultimate aim is nothing"

Despite the existence of some different opinions among scholars, issues concerning the publication of source code are not likely to be the most relevant obstacles, which the limitations existing in article 6 could pose to open source projects. Problems that are much more relevant could concern the decentralized and informal process of information exchange that precedes the writing of actual source code. This is why I used, as a title for this paragraph, Eduard Bernstein's quote about the importance of the "movement" above the "aim". Indeed, what characterizes open source projects is not only the disclosure of the source code of the final product, but the community-based effort during the realization of the project, which is constantly ongoing and which is connoted by an intense sharing of opinions, drafts and information in general. Moreover, the relationships among various contributors are typically far from being formal and access to various pieces of information is necessarily quite free and uncontrolled.

Now, think about article 6.1 that states that decompilation can be performed, provided that various conditions are met, including that decompilation is "performed by the licensee or by another person having a right to use a copy of a program, or on their behalf by a person authorized to do so". In that context, another potential obstacle to FLOSS reverse engineering is whether the reverse engineer must be the same person or organization that creates the independently developed piece of software or act in behalf of such a person/entity. Again, the need for a clear link – *ex ante* – between the reverse engineer and the developer is something obviously existing (and likely superfluous to mandate) in the traditional commercial software industry, but it is an additional (even if likely avoidable) obstacle in the decentralized and informal world of open source development. Referring to the case studies I described (see § *2.2. Project using (also) decompilation*), in Europe and according to the previous interpretation of the Software Directive, probably there could be no cooperation between TinyKRNL and ReactOS, not even if TinyKRNL developers used clean reverse engineering and/or limited themselves to releasing technical specifications obtained through their decompilation activity. In fact, in order to enjoy article 6's exception, TinyKRNL developers could not limit themselves to studying and learning or to replicating functionally some small pieces of Windows; they should realize an independently developed software (incidentally, does this mean that the software must be more

LGPL license. This decision determined the creation of an open source community around the project, which is now known as OpenOffice. See the Wikipedia for more details (http://en.wikipedia.org/wiki/OpenOffice.org).

than a DLL library working as one of the original Microsoft libraries? should it have any autonomous function?) and they should not share the obtained information as information concerning Microsoft's software, but only as information concerning their own independently realized work. In other words, they should work as an independent software house, and they would have several problems (and face significant legal uncertainty) if working just as informal members of a community, trying to learn more and – incidentally – contributing their knowledge participating in a collaborative process.[187]

Finally, and apart from educational projects like TinyKRNL, think about a normal project, needing to achieve perfect vertical interoperability in a "sensible" field, in which the operating system developer may not disclose all necessary interoperability information. For instance, a browser competing with Microsoft Internet Explorer or Apple Safari: let me call it "Firefox" for simplicity. Imagine that Firefox' developers discovered some useful APIs in decompiling the platform controller's browser and/or parts of the platform itself. Documenting these APIs, maybe on a shared website or other public platform, could provide a useful service to the entire open source community, in particular to projects like Wine and ReactOS (wanting to achieve interoperability also with Internet Explorer), but also to other developers of applications. At the same time, the discussion of the economics of decompilation suggests that the negative effect of doing so would just be that the excluding power of the incumbents would be reduced and that small potential competitors would become a slightly more real competitive threat. However, article 6 of the Software Directive prevents the reverse engineers from disclosing this information, which is not protected by copyright (as discussed in the first paper of this dissertation) and which – being secret – is not patented. To be fair, there is a possibility of disclosing this information, and that would be the case if (1) Firefox' developers decided to actually make use of the information itself (something that may not always happen, and likely will not for all the information acquired through decompilation) and if (2) using it was actually necessary to connect to the independently created browser (something which is unlikely in several cases, since we are dealing with vertical interoperability between the browser and an existing platform, so that – once this information has been used to connect to the platform – it is normally not necessary to use it to connect the browser with third parties' applications). To be sure, a third party wanting to realize a platform able to communicate with the newly developed browser would need this information, hence publishing the source code of the Firefox browser – as a form of providing the necessary interoperability information to others – would be legitimate, as discussed above. However, this kind of disclosure is particularly imperfect (because instead of disclosing a reconstructed interoperability specification, one has to release a very specific implementation) and also partial (because only the browser's side of the implementation is released, the operating system's side having to be reconstructed again).

4.3. Forbidden, but potentially welfare enhancing, uses of decompilation

Apart from specific conditions being especially problematic for open source developers, there are several uses of software reverse engineering that aim at purposes, which I am tempted to label as "legitimate," but which seem quite clearly to be outlawed by the Software Directive. According to technologists,[188] the sensible "goals" of decompilation could essentially be: achieving direct interoperability; achieving indirect interoperability; understanding why some bugs are arising; security purposes; verifying if a copyright violation is taking place. As already clarified, it would almost never be sensible to use reverse engineering to "copy" someone else's software in a hidden way, at least unless some of the previous goals are also at stake. I also already discussed that direct (vertical) interoperability is surely a legitimate goal under article 6. Pursuing indirect (horizontal) interoperability is arguably allowed as well. Instead, security purposes and the monitoring of copyright violation are likely not legitimate reasons to perform decompilation under European law. The main issue that could be discussed further – even if, as I anticipated, the literature seems to exclude its legitimacy under the Software Directive – concerns error correction.

To clarify what I am discussing, it may be interesting to borrow the following example of error correction from Spoor:

---

[187] Notice, however, that TinyKRNL developers very likely violated copyright law in distributing code obtained through dirty reverse engineering and which was likely to be a derivative work of Microsoft's code (even if some of the characteristics of this projects give them a decent chance of being awarded a fair use exception in the US). However, I would not see any economic reason – apart from a literal reading of article 6, paragraph 2, letter b – to consider their activity as illicit, in case they did share just independently written specifications, detailing the main results of their decompilation project of Microsoft Windows. Nor general copyright principles would mandate the illegality of a similar activity.

[188] Andrew Johnson-Laird, *Software Reverse Engineering in the Real World*, 19 U. Dayton L. Rev. 843 (1994).

> Some years ago, Tulip Computer, the largest PC-compatible manufacturer in the country and one of the largest in Europe, was one of the first to announce a 386, 16 Mhz PC. Towards the end of the development, the company was confronted with the problem that Lotus 1-2-3 would not access on this particular model; instead a screen message accused the user of using a non-authorized copy. Clearly this was a major problem, since a DOS PC on which Lotus 1-2-3 will not run simply cannot be marketed. Tulip suspected the Lotus 1-2-3 copy protection system to be responsible for the problem. According to Tulip, however, for the very reason that its copy protection system was involved, Lotus refused to provide information and merely suggested that the error was due to some mistake on Tulip's side. Eventually, after decompiling the relevant piece of software, Tulip found that the copy protection system indeed caused the problem, as it was based on a carefully timed question-and-response procedure which did not function correctly because the system ran faster than the software anticipated. The problem was cured by an adaptation of Tulip's own microcode, without any change in Lotus software.[189]

Assuming that Tulip legitimately owned a copy of Lotus software, one could ask whether article 5.1 allows decompilation to be performed for the purpose of error correction. I personally consider that this kind of decompilation should be allowed, since the acts referred to in paragraph 1 include "the permanent or temporary reproduction of a computer program" and its "translation, adaptation, arrangement and any other alteration" (hence, potentially also decompilation). And these activities may be performed "where they are necessary for the use of the computer program by the lawful acquirer in accordance with its intended purpose, including for error correction." And making a legitimate copy of the software run on a legitimately acquire PC is surely a quite normal goal of a customer.[190] To be sure, what article 5.1 clearly and undisputedly means is that it is possible to install an acquired piece of software on the hard disk and to load it into RAM (in fact, these acts involve copying and would otherwise be forbidden despite being necessary to make use of the program). But the article seams to imply something more, with the working "including for error correction". In fact, the words "including for error correction" seem to mean that error correction is typically among the "intended purposes" for which a program is acquired and, hence, that it is possible to legitimately "copy" and "transform" the program for this goal. Because of that, I submit that the interpretation allowing reverse engineering for error correction is not only coherent with intellectual property principles and possibly with consumer law,[191] but even with the literal text of the Directive. However, I must stress again that the majority of the literature[192] does not agree with my interpretation, especially because decompilation is considered a special case, completely dealt with in article 6.

To be honest, I also have to acknowledge that my suggested interpretation (i.e. that decompilation is allowed for the purpose of error correction) is quite weak also in the light of some of the implementations of the directive. Indeed, some countries recognized the ambiguity of this provision concerning "error correction" and decided to be more explicit in dealing with it (while others simply copied and pasted it into their national law, leaving ambiguity intact). For instance, under section 50C of the Copyright, Designs and Patents Act 1988 (c. 48) of the United Kingdom (implementing the Software Directive in UK):

> (1) It is not an infringement of copyright for a lawful user of a copy of a computer program to copy or adapt it, provided that the copying or adapting (a) is necessary for his lawful use; and (b) is not prohibited under any term or condition of an agreement regulating the circumstances in which his use is lawful.
> (2) It may, in particular, be necessary for the lawful use of a computer program to copy it or adapt it for the purpose of correcting errors in it.
> (3) This section does not apply to any copying or adapting permitted under section 50A or 50B.

And, since section 50A and 50B deal with reverse engineering (which is clearly defined in the UK implementation as "convert[ing] […] a computer program expressed in a low level language […] into a version expressed in a higher level language"), it is clear that – at least in the UK – the possibility of copying or adapting a piece of software for the purpose of error correction does not extend to decompilation. What kind of copying and adaptations could achieve the purpose of *correcting* (and not simply detecting) errors

[189] SPOOR, *Copyright Protection and Reverse Engineering*, p. 1079.

[190] Notice that I will not discuss this issue from the point of view of consumer law, apart from an hint in the next footnote but this may be another interesting approach.

[191] Preventing a customer from correcting defects in a good that has been sold to him would not be reasonable and no customer would try to do so as long as the seller demonstrates to be willing to correct the bug itself, because of the prohibitive cost of this activity for a single customer – so, when sellers are reasonably available to correct errors themselves, there will be no unnecessary decompilation activity performed by users.

[192] See, for instance, GUGLIELMETTI, *in* Analisi e decompilazione, p. 159: "non è […] consentito compiere atti di riproduzione o di traduzione del codice in maniera separate dall'utilizzazione pratica del programma, e diretti alla sua decompilazione".

without recurring to decompilation (and to a partial reimplementation and/or creation of a specific patch for the piece of software) is not completely clear to me,[193] but – at least – the UK act is unambiguous. Talking about other implementations of the Directive, only Portugal seems to have drafted its law in such a way that additional scope for a more extensive freedom to decompile (with respect to the one explicitly granted in the Software Directive) could be envisaged. In particular:

> Article 6 (2) (a) has not been implemented. Contrary to the Directive it is therefore not ruled out that decompiling acts may be used for goals other than to achieve the interoperability of an independently created computer program. Finally, the implementation of the three steps test requirement is by no means fully compliant with the wording of Article 6 (3).[194]

To conclude, I have to admit that – whatever the correct interpretation of article 5.1 may be – this issue is by and large empirically irrelevant. In fact, "specific contractual provisions" may derogate the provision of article 5.1 (differently from what happens for the provisions of article 5.2, 5.3 and 6)[195] and readers accustomed to standard software end user license agreements know how systematically decompilation is forbidden, apart from the decompilation activities explicitly and imperatively allowed by the applicable law.

4.4. Concluding (critical) remarks about the Software Directive

In conclusion, article 6 requires "raw information", including abstract ideas and methods (like API specifications), to remain "confidential", unless necessary also to interoperate with the independently developed program. I submit that this is an excessive and unnecessary reinforcement of the already powerful protection offered by technology through the secret about source code. In fact, such a protection would go in the direction of creating a quasi-property right.[196] Moreover, the rules spelled out in the second paragraph of article 6, despite being essentially non-binding or irrelevant for the majority of commercial decompilation projects (not disclosing their results to keep their competitive advantage on later comers), would hinder decentralized and collaborative models of development, like the open source one. As a matter of fact, the conditions spelled in article 6.2 seem to have been stated in order to prevent some kinds of abuses (maybe in part unforeseen), but may have the (again unforeseen) effect of especially hindering a new model of software development.

As I described, achieving perfect horizontal interoperability with a complex platform (like an operating system) is an almost unachievable goal. In the past, commercial efforts in this sense invariably failed and the growing complexity of software seems to doom future efforts to failure as well, despite some progresses in reverse engineering techniques. Hence, software incumbents are more and more likely to become or remain dominant, especially if they control software platforms. However, the open source community have some hopes to succeed in displacing them or, more likely, in limiting their abuse of market power posing a competitive threat. Unfortunately for competition, following all the conditions established by article 6 of the Software Directive risks sterilizing several of the most important spillovers of information that could make open source projects the more credible threats to the market power of dominant incumbents. I will draw more conclusions in the final part of the paper, however I may already mention here that – if article 6.2.b and the other limitation to software reverse engineering in general have some reason to exist – then legislators should take into account the possibility of temporarily limiting them. For instance, by saying that these conditions must be respected for no more than 2-5 years[197] after a piece of software is published. Instead, the reinforcement of trade secret offered the Software Directive lasts as long as copyright. *Id est*, for all practical purposes in the field of software, forever.

5. Decompilation in the US

In the US, software decompilation is governed by the general and flexible principles of fair use. For a description of this doctrine, I remand to the first paper of this dissertation (§ 4.5 and its subsections). As

---

[193] In fact, I suspect that this provision is kind of sterilized remnant of previous proposed drafts of the Directive, where error correction was listed among the legitimate reasons to recur to decompilation.

[194] COM/2000/0199 (*supra note* 184).

[195] See *supra* note 151.

[196] What is happening here is similar to what has been recently observed by Ghidini and Falce about the recent Italian legislation about trade secret. GUSTAVO GHIDINI & VALERIA FALCE, *Recent developments in Italian regulation of trade and industrial secrets: A patent contradiction of the patent regime?*, paper presented at the 3rd Annual Workshop on the Law and Economics of Intellectual Property and Information Technology, 5-6 July, 2007, Queen Mary, University of London (July, 2007).

[197] See REICHMAN, *Legal Hybrids* and SAMUELSON, et al., *A Manifesto*.

already mentioned, on the one hand US norms have the advantage of being more flexible, and likely able to accommodate any of the aforementioned "legitimate" uses of decompilation, from error correction to the monitoring of copyright violations. And, for sure, an extensive list of precedents seems to guarantee the legitimacy of software reverse engineering, if the goal is to achieve interoperability, both vertical (see, in particular, Sega v. Accolade)[198] and horizontal (see, in particular, Sony v. Connectix).[199] On the other hand, recurring to such a flexible doctrine as fair use and basing the expectations of undertaking on facts-intensive precedents reduce legal certainty in the US, with respect to the European situation.

Several authors have already discussed the law & economics of software reverse engineering in the US.[200] For the purpose of the paper at hand, I will just stress a couple of relatively neglected issues, which may in some way connect to the previous discussion about open source decompilation efforts and to the European legal setting. These two topics are the – real or perceived – favor clean-room reverse engineering (which could disfavor open source developers) and the legal rules governing technological measures of protection established by the Digital Millennium Copyright Act (DMCA),[201] where decompilation is allowed, but purpose-bound to the achievement of interoperability (with an EU-Software-Directive-inspired approach).

5.1. The "clean room" process

Theoretically, a single programmer could develop his own software – needing to be interoperable with an existing one – realizing all needed activities, from the decompilation of the original software to the (more or less explicit) realization of an interoperability specification and the actual writing of the new software embedding a compatible implementation. Nevertheless, when a decompilation activity is needed in order to acquire an interface specification, it makes software engineers very familiar with the decompiled software, so that they risk – more or less unconsciously – repeating several of the logical passages chosen by the original developers and ultimately risk realizing a software which is very similar to the original one, also from the point of view of the protected expression (including the actual lines of code). Moreover, the fact of having had access to the decompiled implementation is an advantage for plaintiffs alleging copyright infringement and a problem for defendant wanting to use a defence of independent creation.[202]

For the aforementioned reasons, the software industry has developed a productive process know as "clean room process", reducing the likelihood of being charged of copyright violation or – at least – improving the possibility of success in front of courts. In this process, two separate teams of developers are in charge of the reverse engineering project and of the re-implementation work. In practice, the team actually decompiling the original software works in a "dirty room" and doesn't realize any original software: the goal of this team is simply to produce documentation describing the requisite that any piece of software should respect in order to be interoperable (compatible) with the decompiled one. In other words, only an "interface specification manual" will exit from the dirty room, and not the decompiled code or other materials which could be

---

198 Sega Enterprises Ltd. v. Accolade Inc., U.S. Court of Appeals, Ninth Circuit, October 20, 1992, 977 F.2d 1510: "We are asked to determine […] whether the Copyright Act permits persons who are neither copyright holders nor licensees to disassemble a copyrighted computer program in order to gain an understanding of the unprotected functional elements of the program. In light of the public policies underlying the Act, we conclude that, when the person seeking the understanding has legitimate reason for doing so and when no other means of access to the unprotected elements exists, such disassembly is as a matter of law a fair use of the copyrighted work."

199 Sony Computer Entertainment, Inc. v. Connectix Corporation, U.S. Court of Appeals, Ninth Circuit, filed Feb. 10, 2000, 203 F.3d 596: "Video game system manufacturer commenced copyright infringement action against developer of emulator software that enabled manufacturer's games to be played on computers rather than only on manufacturer's console. […] The Court of Appeals […] held that: […] intermediate copying of BIOS that was necessary to access unprotected functional elements constituted fair use […]".

200 See, in particular and also for further references, SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering*, ; C. R. MCMANIS, *Intellectual Property Protection and Reverse Engineering of Computer Programs in the United States and the European Community*, 8 Berkeley Technology Law Journal, 25 (1993). See also JEFFREY A. ANDREWS, *Comment, Reversing Copyright Misuse: Enforcing Contractual Prohibitions of Software Reverse Engineering*, 41 Hous. L. Rev., 975 (2005).

201 *See* JAMES L. DAVIS, *Is Interoperability just for Those Who Can Hack It? The Application of the DMCA Interoperability Exceptions in the Consumer Electronics Industry*, 2005 University of Illinois Journal of Law, Technology, and Policy, 141 (2005).

202 In fact, the protection offered by the "clean room" process may be higher in common law jurisdictions, where the defence of independent creation is clearly available. Such a defence is not always clearly available in some civil law countries, including Italy, where some scholars and courts seem to favor an "objective" notion of novelty. See also the *General Introduction* to this dissertation, footnote 32 and accompanying text.

considered as "derivative works"[203] of the original implementation. Inside the "clean room", developers without any access to the original decompiled software – apart from the interface documentation produced by the first team – will develop "from scratch" the new software, those similarities with the original implementation will be due only to the interoperability requirements described in the "interface specification manual".

If a clean room process is followed, it will be easier to attribute to technical and logical constraints (or programming conventions) any similarities between the two pieces of software.[204] Moreover, the clean room process may shield developers from some kind of "traps" that the original programmers could (legitimately) put in their software in order to individuate advanced copying through reverse engineering (or leave them by accident): in particular, frequently evidence of copying is deduced from the presence of similar mistakes, useless declaration of variables and other similarities which would be very unlikely, without direct copying, and difficult to justify (without resorting to statistically improbable similar mistakes). The clean room process, imposing the justification of the need to reproduce any specific line of code (otherwise the specification manual should not contain them at all) may reduce the likelihood of unconsciously falling in these traps.

The main limit of the clean "room process" is the fact that it may be difficult (or almost impossible) to implement in some non-traditional organizations. For instance, a (more or less) decentralized network of developers (like the one behind an open source project) may not be easily compatible with this model of software development.[205] More generally, the clean room process may be considered as inefficiently burdensome (in terms of human resources and time).

Because of the costs and inefficiencies potentially linked with the actual implementation of a clean room process – and even if such a procedure, if adopted, may surely give some advantages to a firm charged with copyright infringement – I suggest that developers looking for interoperability should not be obliged (nor *de facto* induced by legal rules) to actually put in place clean room reverse engineering. Courts should just mentally decompose any API reimplementation in two stages: the creation of a specification manual in the dirty room; and the reimplementation in the clean room. To find a violation, the reimplementation should contain some protected expression, but the real problem must have originated in the (real or virtual) dirty room: something must have come out of the dirty room that should not have. Otherwise, the copied expression has been copied out of necessity, because of a merging of ideas and expressions. To be sure, I think that this is already the case in the US, if a correct application of the fair use test is performed, however, a clear cut safe harbor for software reverse engineering would reduce legal uncertainty, without reducing the available level of protection for copyright holders, apart from the fact of more clearly shifting on them the burden of proof for suspect copyright infringements.

5.2. Critique of purpose-bound exception in DMCA

The (in my opinion, unfortunate) purpose-bound approach to decompilation, adopted by the EU Software Directive, seems to have inspired the legislators on the other side of the Atlantic.[206] Indeed, according to some commentators:[207]

> A recently prevalent assumption that 'to better promote the industry, greater protection of the industry's products are necessary' has led to most striking legislative actions: the Digital Millennium Copyright Act (DMCA) and the Uniform Computer Information Transactions Act (UCITA). As a result of the current interplay between copyright law and this legislation, uncertainty with respect to reverse engineering has been compounded. These laws undermine the copyright balance by unreasonably narrowing the scope of 'fair use' rights and likely bringing about anticompetitive effects in the market. For example, anti-circumvention provisions within legislation are not subject to the well-established fair use copyright defense. Thus, even legitimate reverse engineering, other than that performed for interoperability

[203] Obviously ideas and technical rules are "derived" from the reverse engineering process, but they are not "derivative works" in the copyright law sense. In a similar way, a critical essay about a books must "derive" from the reading of the book, but it is also a completely new original intellectual creation and not a derivative work.

[204] The clean room procedure may also be usefully implemented in order to credibly exclude third parties from accessing the decompiled code, which may be useful in order to fully respects the requirements of article 6 of the European Software Directive.

[205] Nevertheless, several open source projects adopted other procedures and complex auditing systems in order to achieve similar results (see the case of ReactOS, described above).

[206] On a comparative perspective, I suggest that this is an interesting evidence of the fact that the emulation of copyright norms does not always flow from the US to the UE, but there is – more likely – a simple race to imitate protectionist approaches.

[207] SEUNGWOO SON, *Can Black Dot (Shrinkwrap) Licenses Override Federal Reverse Engineering Rights?: The Relationship Between Copyright, Contract, and Antitrust Laws*, 6 Tulane Journal of Technology and Intellectual Property, 63 (2004), p. 64.

purposes, which circumvents a security measure solely for the purposes of accessing uncopyrighted functional elements would be condemned.

In other words, US law has implemented a "purpose-bound" interoperability exception in the field of the legal protection of technological measures (of protection of copyrighted content[208]). Indeed, there are just seven exemptions to Section 1201(a)(1)(A)[209] of the DMCA[210] and the one concerning interoperability is strikingly similar to article 6 of the European Software Directive:

> (f) Reverse Engineering- (1) Notwithstanding the provisions of subsection (a)(1)(A), a person who has lawfully obtained the right to use a copy of a computer program may circumvent a technological measure that effectively controls access to a particular portion of that program for the sole purpose of identifying and analyzing those elements of the program that are necessary to achieve interoperability of an independently created computer program with other programs, and that have not previously been readily available to the person engaging in the circumvention, to the extent any such acts of identification and analysis do not constitute infringement under this title.
> (2) Notwithstanding the provisions of subsections (a)(2) and (b), a person may develop and employ technological means to circumvent a technological measure, or to circumvent protection afforded by a technological measure, in order to enable the identification and analysis under paragraph (1), or for the purpose of enabling interoperability of an independently created computer program with other programs, if such means are necessary to achieve such interoperability, to the extent that doing so does not constitute infringement under this title.
> (3) The information acquired through the acts permitted under paragraph (1), and the means permitted under paragraph (2), may be made available to others if the person referred to in paragraph (1) or (2), as the case may be, provides such information or means solely for the purpose of enabling interoperability of an independently created computer program with other programs, and to the extent that doing so does not constitute infringement under this title or violate applicable law other than this section.

Quite interestingly, the previous provisions contained in the DMCA – which, in the field of interoperability look to have been copied and pasted from the European Software Directive – have been described by several authors as "giv[ing] rise to a 'paracopyright,' in which the new access right is designated by a closed set of exemptions." In this new legal setting, even cases in which the "post-circumvention behavior is not a copyright infringement, it might still violate the Anti-Circumventing rule. "[211] And that seems to be hardly acceptable to several American scholars, since "[r]ather than target[ing] the small subsection of copying that is infringement, it targets all copying."[212] However, readers will notice that this "paracopyright" is precisely what the European Software Directive established about decompilation in 1992.

Moreover, "[t]he DMCA is overbroad in that it prohibits the rights of manufacturers to address non-computer software based interoperability issues".[213] In particular, a problematic case is the one in which

---

[208] To be rigorous, one should qualify this statement: technological measures of protection are actually able to protect any kind of content, copyrighted, copyrightable or not protected at all. In fact, much of the criticisms to the DMCA and similar legislative acts implementing the 1996 WIPO Copyright Treaty (as the Directive 2001/29/EC of 22 May 2001 on the harmonisation of certain aspects of copyright and related rights in the information society) pivoting around exactly the fact that technological measures of protection do not (and cannot) respect the boundaries of copyright law (for instance, allowing fair uses and other exceptions and limitations). These topics are largely outside the scope of the present paper. However, as an aside, I would like to mention that I personally sympathize with these arguments, but I think that it should also be recognized that – as long as content can be accessed – no measures of protection could prevent users from deriving inspiration from a work or even quoting or reusing it (maybe just in the form of an analogical copy). For instance, critiques of the DMCA mentioning the absence of an exception for "parody" or "quotations" are simply overbroad: as long as a book or movie can be read or watched a parody can be created and notes to quote a given passage can be taken. And surely no digital copying is needed (not even for quotation, in the form in which it is usually understood). Obviously, much more serious problems could arise in other cases (for some compelling ones, *see* JOSHUA SCHWARTZ, *Thinking Outside the Pandora's Box: Why the DMCA Is Unconstitutional under Article I, § 8 of the U.S. Constitution*, 10 Journal of Technology Law and Policy, 93 (2005), in particular at pp. 130—135 for issues involving hardware interoperability).

[209] "Violations regarding circumvention of technological measures- (1)(A) No person shall circumvent a technological measure that effectively controls access to a work protected under this title."

[210] See DAVIS, *The DMCA Interoperability Exceptions*, . See also ANDREA OTTOLIA & DAN WIELSCH, *Mapping the Information Environment: Legal Aspects of Modularization and Digitalization*, 6 Yale Journal of Law and Technology, 174 (2004), in particular text accompanying footnote 311—312. The exceptions concern: "(a) Non-profit 'shopping' privilege, (b) legitimate law enforcement/national security, (c) necessary program interoperability, (d) legitimate encryption research, (e) protection of minors toward harmful material, (f) protection against collection of personal data (surveillance without notice), and (g) computer security testing."

[211] OTTOLIA & WIELSCH, *Legal Aspects of Modularization and Digitalization*.

[212] SCHWARTZ, *Why the DMCA Is Unconstitutional*, p. 96.

[213] Id., p. 130.

developers want to achieve interoperability with a certain file format or method of transmission or any other protocol "enveloping content" (i.e. any other technology which is not meant just to generate functional effects, but also to "vehicle" or "contain" aesthetic creations). In reality, whenever it is possible to use the information obtained through interoperability also to violate copyright on protected content, then the enjoyment of the interoperability exception becomes very much disputable. And reverse engineering a technological measure of protection – in particular if the information obtained is made available to the public (for instance, publishing the source code of the resulting software) – is an activity which is very likely to have potential infringing uses (or – at least – to significantly facilitate them), as in the following example:

> In Universal City Studios v. Corley,[214] a suit brought by eight motion picture studios (and not even one private citizen copyright holder), plaintiffs sued a consumer who cracked the Content Scrambling System for DVDs and reverse engineered a program that would allow users to play DVDs on systems using the Linux operating system. The Defendant argued that he was specifically exempted by the DMCA exception that allows reverse engineering for computer system interoperability. However the Second Circuit held that this defense was not good enough because the same technology could be used by a non-Linux user to circumvent technological copyright protections. The technology had the possibility of being misused with respect to a protected work, so the circuit court held that the technology was proscribed.[215] […]
> The Corley court's bottom line was, if technology falls under a DMCA exception and has the added effect of circumventing a technological measure, then two provisions of the DMCA are in conflict, and preventing the violation under § 1201(a) takes priority.[216]

Given these precedents, it becomes natural to ask oneself whether it is possible to use the DMCA to keep full control on technologies such as the Trusted Computing Platform and avoid "unofficial" compatibility of any kind, *de facto* foreclosing the software market to several small producers, with particular concern about open source projects. The aforementioned Trusted Computing Platform uses a combination of hardware keys and identifiers in order to make sure that no malicious software is run on a given computer. Some commentators in the open source community think that it may be easily misused in order to foreclose interoperability and asked Microsoft to pledge not to do so,[217] in analogy with what the software house did with its Open Specification Promise.[218] Similar fears should also be mitigated by the behavior of some US courts. In principle, the anti-copying provisions of the DMCA could be (ab)used in order to reinforce market power and foreclose complementary markets, but US courts demonstrated to be able to resist to similar (mis)uses. In particular, in 2004, the Sixth Circuit decided the Lexmark v. SCC case.[219] In that case, Lexmark failed to use the DMCA in order to prevent competitors from producing interoperable products, such as toner cartridges protected by a manufacturer-specific chip embedding an authentication sequence, verified by Lexmark's Toner Loading Program.

In this case, the defendant

> SCC sells its own microchip—the 'SMARTEK' chip—that permits consumers to satisfy Lexmark's authentication sequence each time it would otherwise be performed, i.e., when the printer is turned on or the printer door is opened and shut. SCC's advertising boasts that its chip breaks Lexmark's 'secret code' (the authentication sequence), which 'even on the fastest computer available today … would take years to run through all of the possible 8-byte combinations to break.[220]

In particular, the Lexmark case clarified that it is appropriate to distinguish between the market for the copyrighted work – the technical protection of which is legally reinforced by the DMCA – and other complementary markets – the access to which should not be foreclosed using the copyrighted work as a

---

[214] 273 F.3d 429 (2d Cir. 2001). For a similar case, see also 321 Studios v. Metro Goldwyn Mayer Studios, Inc., 307 F. Supp. 2d 1085 (N.D. Cal. 2004).

[215] SCHWARTZ, *Why the DMCA Is Unconstitutional*, 108—109.

[216] Id., 109.

[217] See Michael Tiemann, *What Microsoft can do for Open Source*, July 25, 2008 (available at http://opensource.org/node/352; last visited July 27, 2008) and Matt Asay, *A prayer for Microsoft*, July 27, 2008 (available at http://news.cnet.com/8301-13505_3-10000366-16.html?tag=blogFeed last visited July 28, 2008).

[218] See *Microsoft Open Specification Promise*, published: September 12, 2006 and last updated: July 25, 2008 (available at http://www.microsoft.com/interop/osp/default.mspx; last visited July 28, 2008). See also Matt Asay, *Microsoft opens up its Open Specification Promise*, July 25, 2008 (available at http://news.cnet.com/8301-13505_3-10000124-16.html; last visited July 28, 2008).

[219] Lexmark International, Inc. v. Static Control Components, Inc., U.S. Court of Appeals, Sixth Circuit, filed Oct. 26, 2004, 387 F.3d 522. For a detailed and commented description of the case, see JACQUELINE LIPTON, *The Law of Unintended Consequences: The Digital Millennium Copyright Act and Interoperability*, 62 Washington and Lee Law Review, 487 (2005), p. 499—510.

[220] 387 F.3d 522, 531 (quoting the D. Ct. Op. 103, at 19).

pretext. More generally, and coherently with what I suggested in the first paper of this dissertation and in the present one, the court clarified how the fourth step of the fair use test should be applied:

> With respect to the fourth factor—the effect of the use on the value of the copyrighted material—the relevant question likewise is whether the infringement impacted the market for the copyrighted work itself. […] Here, the district court focused on the wrong market: it focused not on the value or marketability of the Toner Loading Program, but on Lexmark's market for its toner cartridges. Lexmark's market for its toner cartridges […] may well be diminished by the SMARTEK chip, but that is not the sort of market or value that copyright law protects. […] Lexmark has not introduced any evidence showing that an independent market exists for a program as elementary as its Toner Loading Program, and we doubt at any rate that the SMARTEK chip could have displaced any value in *this* market.[221]

In other words, what Lexmark found valuable in its Toner Loading Program was the possibility of practicing price discrimination, using the sales of its own toners as a monitoring tool for the "intensity of usage" of each customer's printer.[222] As frequently happens for software code necessary to obtain interoperability, its value is not in the creativity or technical complexity of the code itself, but in a competitive strategy enabled by the (absence of) interoperability. In this case – as in other discrimination and two-sided strategies briefly discussed in the first paper (§ 9.3) – it may be the case that some consumers benefited from this kind of differentiation (typically, light-users having the possibility of buying lasers printers for a lower price, indirectly subsidized by heavy-users). However, the court did not think that allowing such a strategy was among the purposes of the copyright act. The court also recalled that similar principles had been applied in Sony v. Connectix:[223]

> Sony understandably seeks control over the market for devices that play games Sony produces or licenses. The copyright law, however, does not confer such a monopoly.

Hence, the "relevant" market for a fair use analysis is not at all the generally relevant market (the competition policy relevant market, for instance) or any market that could be somehow controlled controlling the copyrighted good. When fair use is considered, what matters seems to be the market in which the up-front sunk cost of expression has been supported. And that is consistent with the view of copyright just as a tool to prevent free riding on the costs of expression, trying to avoid any market power concerning ideas. That having been said, the facts of Lexmark case offered the possibility of rejecting Lexmark's claims on the basis of arguments including the likely non-copyrightability of Lexmark's small lock-out software,[224] so that the precedent did not look as clear as it could have about the possible (ab)uses of the DMCA in order to prevent interoperability. This is why Judge Merritt decided to write a concurring opinion, taking the following position:

> I write separately to emphasize that our holding should not be limited to the narrow facts [of this case]. We should make clear that in the future companies like Lexmark cannot use the DMCA in conjunction with copyright law to create monopolies of manufactured goods for themselves just by tweaking the facts of this case: by, for example, creating a Toner Loading Program that is more complex and 'creative' than the one here, or by cutting off other access to the Printer Engine Program. The crucial point is that the DMCA forbids anyone from trafficking in any technology that 'is primarily designed or produced for the purpose of circumventing a technological measure that effectively controls access to a [protected] work.' 17 U.S.C. § 1201(2)(A). The key question is the 'purpose' of the circumvention technology. The microchip in SCC's toner cartridges is intended not to reap any benefit from the Toner Loading Program—SCC's microchip is not designed to measure toner levels--but only for the purpose of making SCC's competing toner cartridges work with printers manufactured by Lexmark.

---

[221] 387 F.3d 522, 544-545.

[222] This is essentially a variation of the traditional strategy of "giving away the razor", to make money selling the blades.

[223] 203 F.3d at 607.

[224] This is not relevant for the paper at hand. However, the copyrightability of the aforementioned small software may be seen as related to the copyrightability of interface implementation. In fact – on this point – I dissent from the majority's opinion in Lexmark and agree with Judge Feikens partly dissenting opinion: "I feel that the record could support a finding that there was enough original expression in the Toner Loading Program to qualify it for copyright protection. Second, although I agree that the district court erred in its factual findings supporting the conclusion that the Toner Loading Program was not functioning as a lock-out code, I feel the record offers support for the proposition that it is possible and practical for competitors to make toner cartridges that function with the printer without copying the Toner Loading Program, and therefore, I would remand that issue to the district court to make a determination in the first instance. Third, although I agree with my colleagues that the district court erred in applying the law of the doctrine of merger and scenes a faire, I would apply the doctrines in this case differently.".

Indeed, the main purpose of the DMCA is to protect creative works against piracy, not to create technological monopolies and incompatibility.[225] Hence, the Lexmark precedent affirms that the DMCA provisions cannot be used, not even indirectly, to prevent interoperability. However, it must be noted that nothing in the case – and it could not have been otherwise, given the clear text of the DMCA – prevents firms from directly using encryption and various forms of Digital Rights Management in order to prevent interoperability with third party's products. But these kind of protections are only as strong as their technology. In fact, the Lexmark precedent clarified that, even if the decompilation exception of the DMCA applies only to software, the DMCA itself cannot be abused in order to prevent other kinds of interoperability.

Unfortunately (from my point of view), ideas, methods and principles embedded in software code protected by technological measure of protection are also legally protected by the DMCA, unless they are necessary to achieve interoperability. The only way to avoid that – apart from a legislative change – would be that the Supreme Court decided that the legal protection of non-patented ideas and information offered by the DMCA violates the copyright clauses of the US Constitution, but this is hardly a likely case. Anyhow, I do not think that this kind of protection could be described as patent-like: in fact, independent creation remains available to everybody, under general copyright principles. Indeed, the only field in which actual access to the original implementation is strictly necessary in order to achieve a technological result – to my knowledge – remains interoperability. Hence the DMCA purpose-bound exception, as its European counterpart in article 6 of the software directive, is arguably unnecessary and slightly detrimental to social welfare and innovation, but I will not overstate its scope saying that it offers a patent-like protection to any kind of software innovation.[226]

5.3. A more general critique: *per se* legality would be better

Despite the fact that no US court ever stated that the clean-room decompilation process is the only legal way to practice software reverse engineering, software developers seem to think that this process is an almost essential pre-requisite to having an acceptable degree of legal certainty while performing software decompilation.[227] In fact, this is not surprising, given the low degree of predictability of fair use cases in general and the various critiques encountered by some US precedents, in particular by the ones allowing for the achievement of horizontal interoperability (see § 7 of the first paper and the critique of Prof. Weiser). For this reason, it would be useful to clarify in some way – possibly with a legislative act – that the clean-room process is not a requirement for a finding of legitimate fair use in software decompilation.

In general, there are sound reasons for criticizing the choice of recurring to a complex analysis as the fair use test to decide whether a decompilation effort is legitimate or not. Such a test would be a perfect tool, if applied by a consensus of enlightened lawyers and economists with a good technological background and mentored by engineers, but one cannot expect that this will be the case. Hence, *per se* legality of software reverse engineering should be considered, limiting the role of courts to decide *ex post* if the use that has been made of the acquired information violated copyright. In particular, courts should verify if derivative works of the reverse engineered software are distributed to third parties (notice that I mean "derivative works" in the copyright law sense of the words: independently written interface specification documents, for instance, are not derivative works). Moreover, one should check whether there have been copyright infringements during the reimplementation of discovered ideas and methods (in the field of interoperability, that means verifying whether interface specifications are re-implemented without technically unnecessary reproductions of the original implementation).

Fair use has been a very useful tool in the hands of US courts in order to deal with software reverse engineering, in the absence of any statutory provision about this technical activity. However, now that courts have created a consistent body of precedents, confirming the legality of software reverse engineering in various cases, legal certainty would be increased by a clear-cut safe harbor for decompilation. Such a safe harbor would not likely entail major costs. Indeed, nobody was ever able to argue in a convincing way that software decompilation is a tool able to create market failures. To the contrary, according to the suggested

---

[225] Or, as the courts puts it in Chamberlain v. Skylink (381 F.3d 1178), at 1193—1194: "The anticircumvention provisions [of the DMCA] convey no additional property rights in and of themselves; they simply provide property owners with new ways to secure their property."

[226] Some authors more or less explicitly argued that, without allowing for decompilation, copyright law would *de facto* attain results which are reserved to patent protection, granting exclusive rights other several kinds of functional results. *See* DENNIS S. KARJALA, *Copyright Protection of Computer Documents, Reverse Engineering, and Professor Miller*, 19 University of Dayton Law Review, 975 (1994).

[227] See above § *2.2.2. ReactOS and TinyKRNL.*

analysis of the economics of decompilation, sketched in the paper at hand, there could even be reasons to stimulate or incentivize software reverse engineering. Finally, the most likely "victims" (in terms of slightly reduced profits and market power, more than likely exclusion from the market) of decompilation would be incumbents with high market power, like Microsoft.[228]

As far as the DMCA is concerned, I will not discuss here the reasons for which I consider the legal reinforcement of technological measures of protection most likely to be ineffective and inappropriate. This topic has been addressed by a plethora of scholars[229] and would require an independent research. However, I have to mention at least that, in cases like Universal City Studios v. Corley, where there is a tension between the DMCA interoperability exception and the possibility of circumventing a technological measure, it would be appropriate to make clear at least how the two interests should be balanced. Indeed, when interoperability is at stake, I think that it should be clarified that the prevention of any risk of violation cannot automatically be considered as a prevailing interest. To reason about that, one must be aware that, from the technical point of view, secrecy about the working of a technological measure of protection is not a necessary precondition for its efficacy. Actually, there are open source implementations of various technological measures of protection and various anti-copyright activists expressed their worry about these systems, precisely because their openness increases the pace at which they may become very robust and effective.[230] It is a fact that open implementation of digital rights management systems are possible. And, despite the likely well-grounded worries of people fearing that these implementation could work "exceedingly well", I think that the possibility of implementing DRM systems on various software and hardware platforms, interoperable among them, may reduce the excluding power of these technologies. Thus, I would propose to think about norms that could incentivize the existence of open DRM systems and favor interoperability among technological measures of protections. That could be done establishing that – if a technological protection measure is so badly designed that knowing its specification makes it ineffective or easy to circumvent – then it should not be considered as "effective" in the sense of section 1201 of the DMCA and, as a consequence, its actual circumvention should not be prosecuted. In order to prevent a possible objection, notice that – to moderate some potential side effects of such an approach – it could be possible to require reverse engineers who discover weaknesses in closed-source technological measures of protection of third parties to inform the original developers and give them a – limited (and possibly statutorily determined) – amount of time to correct these weaknesses before disclosing them to the public through an interoperable open implementation of the DRM system.

Let me also borrow something more from the case study of the Wine project, in order to show that interoperability with the open source world of technological measures of protection is possible, without facilitating the circumventions of these measures:

> Many software vendors include copy protection in their applications (mostly in games). For the most part, copy protection focuses on establishing that the user has an original disc (i.e. a CD or DVD as supplied by the manufacture and not a copy supplied by someone else).[…]
> In an effort to make copy protection more effective (i.e. resistant to cracks), the methods used by many copy protection products have become complex, difficult to understand (obfuscated), and hard to debug. […]. To support copy protection Wine developers have to contend with undocumented interfaces, code obfuscation, and maintaining compatibility with *nix[231] security models.[232] […]

---

[228] On top of having high profits, some software incumbents arguably abused of their market power at least in the US and in the EU (but also in Korea and other countries) and these abuses were frequently somehow related to the foreclosure of interoperability. In the US, I refer to the antitrust case that I labelled *Microsoft III*, see *supra* note 15. In Europe, I refer to the *Microsoft IV* case, see *supra* note 177. About the Korean case, see the third paper, footnote 278.

[229] See, for instance and focusing on interoperability issues, LIPTON, *The Law of Unintended Consequences*, ; DAVIS, *The DMCA Interoperability Exceptions*, . See also SCHWARTZ, *Why the DMCA Is Unconstitutional*, .

[230] "Stallman says that the if you accept the proposition that 'open source' is good because it results in more powerful and reliable software, this makes 'open source DRM' worse than proprietary DRM. As he explains – 'If you think that the important thing is for the software to be powerful and reliable, you might think that applying the OS development model to DRM software is a way to make DRM powerful and reliable,' he explains. 'But as far as I'm concerned, that makes it worse - because it's job is restricting you. And if it restricts you reliably, that means you've been thoroughly shafted.'" See Andrew Orlowski, *Lessig, Stallman on 'Open Source' DRM*, The Register, April 15, 2006. Available at http://www.theregister.co.uk/2006/04/15/lessig_stallman_drm/ (last visited June 15, 2008).

[231] *nix is an abbreviation for Unix-like. In this context, it just means that Wine has to "translate" the requests of these technological measures of protection, in such a way that they are compatible also with the Unix-like architecture of the systems on which Wine runs.

[232] WineWiki, Copy Protection, http://wiki.winehq.org/CopyProtection (July 31, 2008).

> As of Wine 0.9.49 Safedisc (copy protection) has had many fixes and most safedisc 2/3/4 games should now work with this and later Wine releases. […] SafeDisc is a CD/DVD copy protection program for Windows applications and games, developed by Macrovision Corporation, aiming to prevent software piracy, as well as resisting home media duplication devices, professional duplicators, and reverse engineering attempts.[233]

As clearly stated on Wine's website,

> Wine cannot and will not break the functionality of these copy protection products. Wine's goal is to be compatible with Windows software including copy protection. Although some would advocate the use of illegally modified or 'cracked' games, Wine does not support, advocate, or even view this as a solution. The use of cracks is considered off topic on the forums, IRC channels, etc and will not be tolerated […].

Indeed, various kinds of copy-protection systems have been reported to work under Wine.[234] And, in most of the cases, their working required Wine developers to understand what kind of controls these systems do and what kind of services Wine has to provide them.

Another very recent case may be worth mentioning. The Norwegian hacker "DVD Jon" (known for having decoded the content-scrambling system used on video DVDs),[235] lately claimed[236] to have been able to reverse engineer the digital rights management system used by Apple's iTunes and that without simply cracking it, but potentially allowing other producers of players (like Apple's iPod) to play legitimately acquired iTunes files:

> […] Norway native and San Francisco resident Jon "DVD Jon" Lech Johansen cracked FairPlay's DRM.[237] Johansen plans to "license" the code to Apple's competitors through his company, DoubleTwist Ventures. Johansen's new program "wraps" songs with code that mimics FairPlay, enabling iTunes to be played on other devices. Yet he intends to steer clear of legal ramifications under the DMCA. Johansen claims that he reverse-engineered FairPlay and that his code avoids DMCA restrictions by adding protection rather than removing it. He argues his actions are covered by the DMCA's reverse-engineering exemption, § 1201(f). Because the law is relatively untested in this area, it remains to be seen whether Johansen and his program are actually in line with the DMCA.[238]

Developments of this case remains to be seen, but – if DVD Jon's technical claims are accurate as it seems – it could represent the most significant test of the DMCA's interoperability provision.

6. Drawing some preliminary conclusions

In this section, I draw some preliminary conclusions and offer some additional elements, which – in my opinion – support the following points: (1) a clear-cut safe harbor for software reverse engineering would be appropriate; (2) software interoperability should be a goal at least as important as the fight against copyright piracy; (3) the open source model of software development should not be hindered by legal obstacles coming from the law governing either decompilation or technological measures of protection.

6.1. A generalized (and not purpose-bound) safe harbor for software decompilation

Comparing my description of the economics of decompilation and the case studies of actual decompilation projects, the idea that software reverse engineering is both costly and time consuming is reinforced. The likelihood that this kind of activity generates market failures in the field of software is very low. Moreover, the first paper of this dissertation showed that copyright law already offers various tools allowing the achievement of interoperability, but also avoiding free riding on the significant cost of expression. Hence, I argue that the determination of the legitimacy of software decompilation should not be based on later uses of

---

[233] Tom Wickline, Wine with Safedisc and GLSL support, on Wine Reviews, November 11, 2007 (available at http://www.wine-reviews.net/news/wine-with-safedisc-and-glsl-support.html; last visited July 25, 2008).

[234] These include: Safedisc 1, 2 and 4, also known as SafeCast; Securom 4; Ring PROTECH. See http://wiki.winehq.org/CopyProtection for more information.

[235] DVD Jon (who was 15 at the time!) was charged with copyright violation, but Norwegian courts acquitted him after having verified that he actually "cracked" his own legitimately acquired DVDs. Evidently, the same could not have happened under the DMCA, also because DVD Jon published its results on the Internet, *de facto* allowig other people to crack the DVD scrambling system. (See f.n. 236.)

[236] See Robert Levine, *Unlocking the iPod*, Fortune Magazine, October 23, 2006 (available at http://money.cnn.com/magazines/fortune/fortune_archive/2006/10/30/8391726/index.htm; last visited July 22, 2008).

[237] FairPlay is the DRM technology used by Apple's iTunes.

[238] DEANA SOBEL, *A Bite out of Apple? iTunes, Interoperability, and France's Dadvsi Law*, 22 Berkeley Technology Law Journal, 267 (2007), p. 288—289.

the obtained (unprotected) ideas and principles (information). On the contrary, decompilation should always be considered as legitimate, while the legality of the aforementioned later uses should be based on the market effect of the use of the obtained "expression". That means that, in the EU, the law should be changed to reflect this principle, essentially abrogating article 6.2.a and 6.2.b of the Software Directive. In the US, no legal changes would be strictly needed. In principle, courts should just be careful in interpreting clean-room reverse engineering as a tool favoring defendants against alleged copyright infringements during the re-implementation stage and not as a necessary step for legitimate decompilation. However, a clear-cut statutorily established safe harbor could reduce legal uncertainty and transaction costs supported by the legal system.

It is important to notice that making software decompilation legal with a clear-cut safe harbour would not condone copyright infringing uses of the obtained information. After all, books are easily understandable, but they cannot be copied and the same holds true for open source software. About that, it may be appropriate to notice that "the exam of case law demonstrates that the vast majority of copyright infringement cases consist in conducts of authentic *duplication*, or quasi-integral reproduction also of the literal code.[239]" Italian cases, quoted by Guglielmetti, frequently show percentages of identical code above 80% or approaching 100%[240]. Moreover, these cases of blatant infringement are normally based on direct access to the original source code or even on direct copying of part of the distributed object code. This kind of parasitism will not be condoned by a safe harbour for decompilation, simply because true economic parasites do not engage in serious reverse engineering, because it is an excessively complex and costly activity. Free riders tend to mask literal (or otherwise automatic) copying. The same author[241] stresses some other problems that should be carefully taken into account, but which are not – once again – related to reverse engineering. The violation of trade secrets and non-disclosure agreement, other forms of unfair competition or unfair business practices, frequently related to former employees' disloyalty and/or breaches of fiduciary duties. In fact, similar issues are behind the vast majority of copyright infringement cases in the field of software, including world-famous US cases, like Computer Ass. V. Altai (where a former employee of CA took with him copies of the source code of various programs and went to work for Altai, the founder of which was another former employee of CA[242]); Cmax/Cleveland v. UCR (where the defendant and the plaintiff where former business partners and the defendant had "requested the source code [of the plaintiff's program] in order to better communicate problems to and request improvements"[243]); Whelan Assoc. v. Jaslow Dental Lab (where the plaintiff and the defendant were business partners and the defendant's program was so clearly derived from the plaintiff's original one that it had been advertised as "a new version" of the former jointly distributed system[244]). In the last two mentioned cases, the actual developers of the infringing code were also inexperienced programmers, essentially learning by doing while copying from the source code of professional programmers; hence – even without being computer experts – it is actually quite easy to imagine that the copying was not limited to general principles and ideas.

In summary, reverse engineering is not the appropriate tool to practice low cost competition. This kind of competition is normally only feasible if one engages in some forms of unfair practices, and there are specific legal tools to deal with this category of behaviours. Hence, both a reverse engineering safe harbour, as suggested in this paper, and a narrow reading of the protection of structural software elements (idea, principles, methods and algorithms, including interfaces abstract specification), as suggested in the first paper, are coherent with a legal paradigm of copyright protection that does not tacitly condone infringement. Finally, notice that – even if, in the future, decompilation became an easier, cheaper and faster process – provisions like article 6.2.a and 6.2.b should be limited in time, introducing just a short blocking period to the disclosure (or even to the use) of the information obtained through decompilation. A quasi-perpetual, quasi-property rule protecting ideas and principles hidden in compiled code is incoherent with general intellectual property principles. A blocking period, if any, should be justified as a quasi-liability rule, established to artificially recreate some lead-time for the incumbent, following the suggestion of Prof. Reichman[245]. The

---

[239] Tentative translation from GUGLIELMETTI, *L'invenzione di software (2nd ed.)*, p. 295-296. See Id., also for additional references.

[240] Id., p. 296 and footnote 94 in particular.

[241] *See* Id., p. 297.

[242] *See* the Discrict Court's Findings of Fact in case 775 F.Supp. 544, with particular reference to section II.E, *Dramatis Personae* (p. 553).

[243] *See* 804 F.Supp. 337, § 10, 19 and 47 in particular.

[244] *See* 797 F.2d 1222, 1227.

[245] REICHMAN, *Legal Hybrids*.

duration of this artificial lead time should be in the order of a few years: about two or three, unless the pace of software innovation slowed down dramatically.[246]

6.2. If they are working only because they are secret, TMPs are not so "effective"

Despite the fact that both the European Union and the United States introduced specific rules to deal with massive piracy performed over the Internet and/or thanks to digital technologies, one should always remember that these norms were not aimed at hindering activities, which – like decompilation – have nothing in common with piracy and are aimed at understanding concepts, spreading ideas, overcoming barriers to entry, increasing fair competition.

The existence of interoperability at the level of DRM systems and other technical measures of protection between software platforms may be a positive element to allow – at the same time – both the possibility of protecting rightholders' right and consumers' rights. It is precisely when TMPs are not interoperable between various platforms that a tension arises between legitimate users and rightholders, and this tension is not in the interest of neither. For instance, think about a copy-protected DVD. Several courts – like the French Court de Cassation in the Mulholland Drive case[247] – established that the consumer does not have a right to copy the DVD on an old VHS cassette in order to play it on a traditional VCR. Frankly, this limitation of consumer's right – as long as it is transparent at the moment of purchase – is not especially worrying to me, because allowing interoperability with an obsolete technology like VHS would require a complete circumvention of the DRM system and the possibility of performing any number of copies and copyright violations starting from the unprotected copy. In other words, asking the rightholder to remove the DRM in such a case, would be equivalent to legally impose the inefficacy of the technological measure of protection. From the point of view of social welfare, I am not very worried by the fact that the old VHS technology is discriminated. However, think now about the possibility of playing on Linux a DRM-protected DVD playable under Windows and/or Mac. As long as the software running under Linux could – technically speaking – play the protected DMR without allowing illegal copies, favoring interoperability should be both in the interest of legitimate users and in the interest of the rightholder, because this increases its potential market. And, in this case, favoring Windows over Linux (or any other platform other another) may create distortions of competition and – in particular – create inertia protecting the existing market leaders - something which is not good for innovation neither, ultimately, for social welfare. If the rightholder does not favor interoperability, the self-help of Linux programmers in decompiling the technological measure of protection to realize a compatible reader is surely in the interest of consumers and – by itself – does not compromise the interest of the copyright holder. If there is no technical way to implement the DRM in an open source way and keeping it safe, the responsibility should be on the rightholder, either to prove that it was actually possible to implement a safe open source version of the DRM or to realize a better DRM (since DRM systems being secure also when they are open source exists).

This approach may seem punitive against closed source producers of technological measures of protection. However – if they take into account this legal setting when deciding their strategies – they would simply be encouraged to realize better DRM systems and to disclose the associated specifications in the first place, avoiding the social waste of decompilation, increasing the availability of platforms on which legally acquired content may be enjoyed. After all, discouraging secret is a tool that has always been crucial in the arsenal of intellectual property and that is crucial in making intellectual property precompetitive (and compatible with antitrust policies). Obviously, such an approach would have the side effect of reducing the feasibility of strategies such as that of Apples with iTunes, where DRM-protected music is arguably given away at cost (or under) and profits are made by selling a proprietary-DRM-compliant device such as the iPod. Nevertheless, should copyright law have the goal of making such a strategy possible? Or, should we simply leave Apple (unless it is found to be a dominant player in this market) to use secrecy to protect the special DRM installed on the iPod, leaving competitors also free to re-implement the same system on other music players?

Here I am not proposing to mandate disclosure of the specifications of any TMP. What I am arguing is simply that secrecy on the working of digital rights management systems should be an instrument available to distributors of copyright protected content, but not a law-protected/incentivized tool. In other words, my suggestion is that economic actors should be free (as a general rule) to put obstacles to the achievement of compatibility with their products (for instance using trade secret), but that the law should not increase the

[246] PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, see id., 2308--2431 .

[247] Cass. 1e civ., Feb. 28, 2006, Bull. civ. I, No. 05-15824. See also SOBEL, *A Bite out of Apple?*, p. 277—278.

natural degree of excludability of information goods, unless this additional excludability is needed as an incentive to the initial investment needed to create the information good. General rules limiting industrial espionage and protecting trade and industrial secrets are a sufficient tool to discourage waste and inefficient activities, but – apart from these rules – I suggest to deal with these problems using other tools – like unfair competition – more than IP. In fact, I suggest using IP only to create incentives to the production of immaterial goods and copyright in particular to allow the (possibility of) recoupment of sunk cost. All the rest – which includes marketing strategies which could be beneficial, subsidies to the producers of complementary products and so on – may be economically and socially useful, but granting a monopoly over immaterial goods in order to make possible these marketing strategies is too risky.

6.3. Coordination with Patent Law

In principle, patents and copyright are two independent forms of protection of computer programs, which can coexist. A certain technical solution – using the vocabulary of the first paper of the dissertation at hand, we could say the "specification of a technical solution" – may be protected by one or more patents, while each of its implementations (in the form of source and object code) is normally copyright protected.

This also means that exceptions to each form of protection could apply only if the allowed uses are not protected by the other legal tools or if there is an exception in both fields. In other words, "experimental use" exceptions covering patents can be enjoyed, but only if this does not violate copyright and *vice versa*.[248] Unfortunately (from the point of view of legal certainty), it has been observed that the experimental use defense may be given "remarkably varying latitude," depending on various legal system or even specific courts.[249] Luckily, if software is not patented as a product, but as a system or apparatus – as frequently happens in Europe (and as was also recommended in the draft European Directive on the patentability of computer implemented inventions) – the "mere creation of copies of the program performed in order to find interfaces could not represent an infringement (not even 'indirect' or 'contributory')".[250] Moreover, reverse engineering will not typically imply any actual use of the patented invention and will therefore enjoy an "experimental use" exception also in the patent field. Hence, legal issues preventing decompilation will arise almost only in cases in which a piece of software is protected as a product or as a method (and not as part of an apparatus or system) and very likely only in cases in which the object of the patent claim is precisely an interface. However, to be sure, in some cases it will not be possible to distribute implementations of what has been discovered during decompilation, if these implementations violate patent law, hence the possibility of decompiling could be emptied of almost any practical meaning (even though reconstructing an approximation of the original source code may provide hints, which could make possible to "invent around" the existing patent).

This implies two things: first of all, the majority of problematic cases will involve pure software patents, i.e. cases in which an interface is the main claim of a patent. Hence, in Europe it is quite difficult to imagine cases in which reverse engineering could be hindered by a patent. Moreover, if the main claim of a patent is an interface, either the patent is not valid – and in this case reverse engineering cannot be stopped – or it is – and in this case re-implementing the specification resulting from reverse engineering would constitute a patent infringement. Hence, in such a case a patent license is necessary and – if this license is obtained – the licensor will typically offer also the possibility of seeing a detailed interface specification without any need of decompiling the original implementation.

Finally, notice that in some countries – for instance, in Italy – where a given interface is necessary in order to implement another software innovation – or even an improved version of the original specification – it may be possible, for the subsequent innovator, to ask and obtain a mandatory license.[251] However, talking again about the Italian case, this obtainment is dependent on the fact that the independently developed software, needing a license for the patented innovation in order to be produced and distributed, represents

---

[248] For a mode detailed legal analysis, *see* GIOVANNI GUGLIELMETTI, *La proposta di Direttiva sulla brevettazione delle invenzioni in materia di software*, 1 Rivista di diritto industriale, 438--463 (2002), 461ff..

[249] See RICOLFI, *Antitrust Antidote*, pp. 356—357. pp. 357 ff..

[250] My translation from GUGLIELMETTI, *La proposta di Direttiva*, , footnote 41, at p. 461.

[251] See in particular GUSTAVO GHIDINI, *Intellectual Property and Competition Law. The Innovation Nexus*, (Edward Elgar. 2006) (already in GUSTAVO GHIDINI, *Profili evolutivi del diritto industriale. Proprietà intellettuale e concorrenza*, (Giuffrè, Milano. 2001)), but also PAOLA A. E. FRASSI, *Innovazione derivata, brevetto dipendente e licenza obbligatoria*, I Rivista di Diritto Industriale, 212--226 (2006).

"an important technical progress of significant economic relevance"[252]. Moreover, such an obtainment may also be conditioned by the more or less burdensome administrative procedure. In any case, I am not aware of any similar mandatory licenses granted in this field in Italy, but this may also be related to the fact that pure software patents are supposed to be essentially inexistent in Europe and are seldom litigated.

6.4. Legislative developments taking into account some of the arguments of this paper

As already hinted when discussing the basic economic model of decompilation, there are several analogies between software and semiconductors. From the technical point of view, there is a significant degree of equivalence between these two tools.[253] Also from the economic point of view, both fields are characterized by similar paths of incremental innovation, economies of scaled and network effects. In fact, also the legal protection of innovations in the field of software and of semiconductors is similar. In fact – despite the fact that semiconductors are governed by an ad hoc regime – this legal tool is largely inspired by copyright. The legal protection of software and semiconductors differs, in particular, in the rules governing reverse engineering. In fact, not only is reverse engineering always allowed, but competitors supporting both the cost of reverse engineering and the cost of developing an improved product receive the possibility of reproducing also the topographies and mask works (i.e. the external form/expression) of the preexisting analyzed semiconductors. (For more details, I remand to Samuelson and Scotchmer[254] and/or Guglielmetti[255]). Such a legal provision makes perfect sense if reverse engineering is costly, incremental innovation is crucial, dominant positions absent or shaky and it legislators want to promote a vibrant competitive market. In fact, as shown by the reasoning about the reverse engineering of software, it is far from sure that decompilation or similar activities lead to market failure (even if, in fact, it could be argued that software reverse engineering is even more difficult than semiconductors reverse engineering and if one should remember that – in the field of software – I always assumed that copyright would, in any case, prevent the literal copying of existing expression, unless technically strictly necessary).

To conclude this brief parenthesis about semiconductors, it must be noted that, according to some commentators, the different legal rules governing software and these products have not been generated by different economic principles governing innovation in these fields. On the contrary, these differences would have been generated by a different timing for the introduction of this legal protection and by the different industrial structures at the time of drafting (in particular in the United States, while the rest of the world followed their leadership both in terms of technological advances and dimension of the market).[256] In my opinion the main difference is not so much that there was more concentration in the software market when copyright was established as a tool to protect computer programs (likely, it was the opposite), but the fact that the "competitive fringe" had very different characteristics in the two markets. On the one hand, in the semiconductors industry, the "fringe" was formed by firms that were relatively small with respect to the market leaders, but of medium average size according to absolute measures. On the other hand, the competitive fringe in the software market was really formed by very small (if not individual) firms. Basic arguments of public choice suggest that the group formed by more individuals with more disperse interests will be the less effective in influencing the choices of legislators. And apparently this is what has happened, giving rise to legal settings in which software incumbents enjoy a much higher level of legal protection of their market power than incumbents in the field of semiconductors. Interestingly enough, today the open source movement, adding ideological (or even philosophical, according to somebody) arguments to the debate, has been able to better represent the interest of a part of this software market competitive fringe. Thus, several instances of the movement have been taken into account by law, and by European legislators in particular (and that is not likely to be completely independent from the fact that the big commercial software houses are mainly based in the US).

---

[252] Article 54, 2, b L.I.. According to several interpretation (see GHIDINI, *IP and Competition Law*,) the "economic relevance" may concern either the profits of the wanna-be licensee and/or the society at large.

[253] In principle, there is an almost complete technical equivalent among the two. See GUGLIELMETTI, *Le topografie dei semiconduttori* for more comments and references.

[254] SAMUELSON & SCOTCHMER, *The L&E of Reverse Engineering.*

[255] GUGLIELMETTI, *Le topografie dei semiconduttori.*

[256] Id..

6.4.1. Failure of the directive proposal on software implemented inventions

One of the first legislative developments that surely took into account the reality of the open source software model, and in a quite spectacular(ized) way, is actually a failed reform: the rejected EU Directive on the patentability of computer-implemented inventions.[257] The legislative history is quite well known, for having been intensively reported by mass media and unprecedently monitored from the Internet.[258] In a few words, the draft Directive experimented several years of debate and numerous conflicting amendments. The final proposal was rejected on 6th July 2005 by the European Parliament by an overwhelming majority (648 to 14 votes, 18 abstentions).[259] Indeed, both sides of the dispute had the possibility of claiming a victory: on the one hand, the EU did not impose on member states any rule concerning the patentability of software; on the other hand, the European Patent Office could continue to issue quasi-software-patents undisturbed. It is interesting to compare the legislative history of this Directive with the one of the decompilation exception of the Software Directive. For the latter, a compromise solution (entailing purpose-bound decompilation under the conditions discussed at length in this paper) has been preferred to either clear-cut solution (i.e. a complete safe harbour for decompilation or a complete ban on it). In the case of the patentability of software implemented inventions, the complete rejection of the draft has been preferred to any compromise solution, considered as inadequate by both sides. Unfortunately, both approaches are prone to legal uncertainty and likely deter innovation and/or competition.[260] For the purpose of the paper at hand, I mention the failure of the Directive in order to stress the new and growing relevance of the open source movement in decisions concerning innovation policy and software in particular. A relevance that was still inexistent at the time of the drafting of the Software Directive, but which is today explicitly recognized:

> The common position, if approved, would have allowed patenting of computer-implemented inventions. This outcome was advocated by big software firms, which argued that patents would encourage research spending and defend European inventions from US competition. On the contrary, the directive was criticised by supporters of 'open source' software, mainly smaller companies, who claimed copyright already protects their inventions and were afraid that patenting would raise legal costs.[261]

Despite the fact of being slightly off-topic in the paper at hand, I want to briefly comment on the issue of software implemented inventions. In fact, there are sound reasons to propose some kind of patent-like protection for software, and highly sophisticated legal reasoning that could be used in order to make this protection coherent not only with the European Patent Convention[262] abound, and are essentially based on the distinction between pure software patents (making no "technical contribution"[263]) and computer implemented inventions.[264] However, there are also significant arguments suggesting that it would be irresponsible to reinforce software patents and make them a main legal tool in this field of technology. Indeed, the main problem of patent protection is that it is not a very friendly tool for small (or even individual) developers, while there is some evidence suggesting that even major players find it more useful as a strategic tool in dealing with other big players and as a barrier to entry and/or legal threats in dealing with the small ones. And, differently from what happened in other technological fields (I am not sure whether without any responsibility for IP law) individual innovators still play a major role in the software field. And

---

[257] 2002/0047/COD.

[258] In particular, several blogs and more traditional websites kept activists and the general public informed about the background of the draft directive and its progresses. Some examples (and additional links) can be found here: http://ciaran.compsoc.com/software-patents.html or http://eupat.ffii.org/ (both last visited August 11, 2008).

[259] Note (06/07/2005 - EP: position, 2nd reading) from the European Parliament's "Legislative Observatory" website, available at: http://www.europarl.europa.eu/oeil/file.jsp?id=219592 (last visited August 11, 2008).

[260] About the cost of uncertainty in the patent system, see JOSHUA S. GANS, et al., *The Impact Of Uncertain Intellectual Property Rights On The Market For Ideas: Evidence From Patent Grant Delays*, NBER Working Paper 13234 (July, 2007).

[261] Note (06/07/2005 - EP: position, 2nd reading) from the European Parliament's "Legislative Observatory" website, available at: http://www.europarl.europa.eu/oeil/file.jsp?id=219592 (last visited August 11, 2008).

[262] See GUGLIELMETTI, *L'invenzione di software (2nd ed.)*, p. 188 (and 166—190 in general).

[263] It is quite telling that in the EPO's brochure "Patents for software? European law and practice" (available at http://www.epo.org/topics/issues/computer-implemented-inventions.html; last visited July 20, 2008) the terms "technical contribution" and "technical character" are mentioned several times, but never defined. Obviously, the definition of these terms is left to the EPO itself and to national courts, but this did not satisfy the European Parliament, which – during the debate and the works preceding the rejecting of the Directive – repeatedly asked for "a clearer definition of 'technical contribution'" (see the note of the European Parliament's Legislative Observatory mentioned in footnote 267).

[264] Ibid.: "[A] computer-implemented invention is an invention whose implementation involves the use of a computer, computer network or other programmable apparatus, the invention having one or more features which are realised wholly or partly by means of a computer program."

favoring concentration is a move that could hardly be reversed, hence one should carefully think about damaging the high level of "diversity" existing in this field. The empirically most relevant argument against software patents is precisely the fact that software innovation is thriving even without them. Unfortunately, nobody has ever suggested a convincing way to measure the "optimal" level of software innovation; hence, it is not possible to say whether the current level – despite being high – is "high enough". However, introducing an additional level of law-backed monopoly in a market where competition – helped by quite weak rules against pure free-riding, in the form of technology copyright – seems to work would likely be irresponsible. In absence of additional convincing evidence, Fritz Machlup's old (but still valid) suggestion seems to go against the introduction of software patents:

> If one does not know whether a system 'as a whole' (in contrast to certain features of it) is good or bad, the safest 'policy conclusion' is to 'muddle through' - either with it, if one has long lived with it, or without it, if one has lived without it. If we did not have a patent system, it would be irresponsible, on the basis of our present knowledge of its economic consequences, to recommend instituting one. But since we have had a patent system for a long time, it would be irresponsible, on the basis of our present knowledge, to recommend abolishing it.[265]

Indeed, a system where "software implemented inventions" are a normal legal tool in the field of software innovation is a completely new system. Instead, making some legal clarity about the impossibility of patenting software in Europe would likely not change any in the existing equilibria in software markets.[266]

As a final note, concerning interoperability, one of the Parliament amendments concerned the introduction of a new article (6a), requiring

> Member States to ensure that licences are available to use a patented computer-implemented invention 'on reasonable and non-discriminatory terms and conditions' when such use is indispensable for achieving interoperability between computer programs and is in the public interest.[267]

Hence, it is reasonable to argue that interoperability was one of the crucial worries of the members of the Parliament opposed to the Directive.[268] And it is similarly reasonable (even if slightly more stretched) to argue that the impossibility of preventing interoperability eliminated much of the appeal of software patents for the coalition favoring them, so that maintaining the status quo of legal uncertainty (allowing some room for legal threats of uncertain strength) was preferred to a Directive clearly allowing patents on software implemented inventions, but also clearly mandating software interoperability under the patent system.

6.4.2. The Loi DADVSI (interoperability among DRM systems)

The failure of the Directive on the patentability of computer implemented inventions was acclaimed as a great victory by the open source community (despite the awareness that patents on software implemented inventions were – and still are – regularly granted by the European Patent Office). But the lobbying of the

---

[265] Machlup, Fritz (1958), An Economic Review of the Patent System, Study no. 15 of the Subcommittee on Patents, Trademarks, and Copyrights of the Committee on the Judiciary, United States Senate, 85th Congress, Second Session (Washington, D.C.: Government Printing Office).

[266] That having been said, the European Patent Convention (EPC) is already quite explicit. It puts "programs for computers" in the same category of non-patentable subject matters as "schemes, rules and methods for performing mental acts, playing games or doing business" and also "discoveries, scientific theories and mathematical methods". In fact, it is well known that the exclusion of all these subject matters just concerns "such subject-matter or activities as such". Hence, it is possible to patent a specific technical solution, with industrial applicability, even if it applies a "scientific theory" or a "mathematical method", or a "computer program". But it should be clear that the patentability of computer programs, simply used to run computers without any industrial application, is excluded by the Convention.

[267] Note (COD/2002/0047 : 20/06/2005 - EP: decision of the committee responsible, 2nd reading) from the European Parliament's "Legislative Observatory" website, available at: http://www.europarl.europa.eu/oeil/resume.jsp?id=219592&eventId=904553&backToCaller=NO&language=en; last visited August 11, 2008). See also RICOLFI, *Antitrust Antidote*, who noticed also that a broad, sector-specific interoperability exception (for the purpose of reverse engineering and to achieve interoperability) could be used to limit the rights conferred by patent law, likely remaining compliant with Articles 27 and 30 of TRIPs.

[268] As a quasi-humoristic, but quite telling note, consider that the main repository of open source projects, SourceForge.net awarded to Wine its 2008 Community Choice Award in the category "Most Likely to Be Ambiguously and Baselessly Accused of Patent Violation". (See http://sourceforge.net/community/cca08-finalists; last visited July 28, 2008). Other finalist projects were frequently related to interoperability and/or reimplementation of commercial technologies, including ReactOS and Mono (open source reimplementation of the .NET client and server applications under Unix-like systems).

open source movement and of small software developers had an even more proactive effect in the context of the French Loi DADVSI.[269]

The law "2006-961 du 1er aout 2006"[270] has been discussed by the French Parliament in a moment in which interoperability was an issue at the centre of the political debate, largely as an effect of the opinion movement created by the rejected Directive on software implemented inventions. For this reason, the main innovations introduced by that law particular concern interoperability[271] between DRM systems. According to the French law, technological measures of protection (TMPs) must not hinder 'the effective application of interoperability', understood as in the Software Directive:[272]

> "Consequently, providers of technological measures must communicate the information essential for interoperability under specified conditions. Technological measures may not go further than what the law permits, namely protecting and guaranteeing an intellectual property right. The Intellectual Property Code seeks to prevent the application of controls restricting access to a work or other protected subject matter beyond the framework of intellectual rights, that is to say the regime of exclusive rights accompanied by exceptions. The law has entrusted a body called the *Technological Measures Regulatory Authority* with the task of ensuring that this is the case."[273]

Clearly, the law tries to avoid interoperability being used as a pretext in order to circumvent DRM systems. But, at the same time, it is also aimed at preventing any use of the legal protection of TMPs to foreclose competition and interoperability, more than to safeguard intellectual property rights. Indeed, the authority established by the Loi DADVSI, has to coordinate with the Competition Council (and both authorities could refer cases to the other body or seek opinions). In any case there are differences between the scope and goals of the activity of the two bodies, since the Competition Council restrains itself to cases in which technological measures may have the effect of strengthening a dominant position (or otherwise violating competition law), while the new law has a much broader application (in a much more specialized field).

In case the information necessary to achieve interoperability (interface and file format specifications) is not disclosed by the manufacturer of the TMP, the Law also establishes a procedure for requesting the disclosure of the source code of the digital rights management system:

> any software publisher, manufacturer of a technical system or owner of an internet service may, in the event of being refused access to information essential for interoperability, ask that the Authority […] [ensure] the interoperability of the systems and existing services […] and obtain from TPM rights holders the information required for [interoperability]. The Authority may compel disclosure of source code and inflict a monetary penalty that is proportional to the damage caused by non-disclosure if the DRM owner refuses to comply.[274]

Obviously, the source code has to be maintained confidential and the French Constitutional Council also ruled[275] that companies must be granted a fair compensation if such a disclosure is required. As far as I have been able to ascertain, no practical application of this norm take place yet.

The Loi DADVSI explicitly treats a special kind of interoperability and does not prevent the ordinary application of norms granting a decompilation exception in light of the Software Directive (in France, Article L.122-6.1). This approach has been criticized by some scholars:[276]

> This cumulative application may lead to a reduction in the situations in which the interoperability introduced by the Law of 1 August 2006 could operate and, above all, it may lead to difficulties. Thus one

---

[269] *See* SOBEL, *A Bite out of Apple?*.

[270] The final version of the law has been significantly modified by the French Senate with respect to the original one, even more innovative, initially approved by the National Assembly. See YVES GAUBIAC, *Interopérabilité et Droit de Propriété Intellectuelle (with en. translation: Interoperability in Intellectual Property Law)*, 211 Revu Internationale du Droit d'Auteur, 91--139 (2007).

[271] The word interoperability is not defined in the law: the reason is likely to be the that defining such a technical concept would have led to rapid obsolescence of the law, but also the fact that the French legislator was approving a law implementing a European Directive and the Software Directive provides a (more or less explicit) definition of interoperability. This definition has been interpreted – according to some commentators (*See* HART, *Interoperability Information*,) in an expansive way – by the European Commission in its Microsoft Decision (2004) and this interpretation has been fully backed by the CFI in the very recent Microsoft verdict of September (17th) 2007.

[272] There is no explicit definition of the concept of "interoperability" in the Loi DADVSI, hence this concept must be interpreted directly recurring to the Software Directive.

[273] GAUBIAC, *Interopérabilité et Droit de Propriété Intellectuelle.*

[274] Loi DADVSI, Article 14. English translation from SOBEL, *A Bite out of Apple?*, p. 275—276.

[275] French Constitutional Council decision no. 2006-540DC, July 27 2006, J.O. 178, para. 41.

[276] GAUBIAC, *Interopérabilité et Droit de Propriété Intellectuelle*, pp. 111-113.

> such case could arise in connection with the communication of information to others, since Article L. 331-7 [introduced by the more recent law] provides that 'The holder of the rights in the technological measure may not require the beneficiary to refrain from publishing the source code and technical documentation for his or her independent and interoperating program unless the holder produces evidence that this would seriously impair the security and effectiveness of the technological measure', whereas Article L. 122-61 lays down, as a condition for interoperability, that the information obtained may not be 'communicated to others, except where necessary for the interoperability of the independently created program'. [277]

According to critiques, in the Loi DADVSI "publication is the rule", while in the Software Directive it was a kind of exception. Actually, as I already discussed, the publication of the source code of a program developed also thanks to information obtained through decompilation was arguably "the rule" also under the Software Directive. However, it is true that the Directive never explicitly states that this is the case. Moreover, the explicit favour of the French legislator for the possibility of disclosing the source code of an interoperable implementation of a TMP is unprecedented and surely worth noting.

In summary, the Loi DADVSI is surely influenced by a new consciousness concerning the open source model of software development, open standard and similar phenomena. That having been said, the "conflict" between the approach adopted in the Software Directive and the one of Loi DADVSI should not be exaggerated. First of all, according to the interpretation of the Software Directive I proposed in the paper at hand, there is a substantial freedom to publish source code of interoperable programs (even if some limitations to the information contained in the comments to the code may exist). Moreover, the two bodies of law deal with information the origin of which is significantly different. On the one hand, in the context of the Loi DADVSI there is no doubt that what is disclosed is not a derivative work of the original implementation (as it may be the case for some intermediate products of decompilation), but simply a technical specification, embodied in a new and "independently created" implementation, having the same (or a very similar and compatible) specification. Hence, from the copyright law point of view, any non-disclosure requirement would be less justified than in the case of the decompilation exception of the Software Directive (where the boundary between protected and unprotected acquired information could have been less clear). On the other hand, in the case of the Loi DADVSI, there could be additional reasons for a confidentiality requirement, because interoperability specifications not directly and independently acquired through decompilation (which is a costly process), but are disclosed by the author.

Overall, I think that it may be reasonably argued that the information concerning the interoperability of DRM systems is much more "sensible" than general information concerning interoperability. For this reason, the new law can be read in the sense of an especially pronounced new sensibility of legislators (or at least of the French one) toward the open source model of software development. Additionally, the fact that the French legislator adopted such an open approach, claiming at the same time to be coherent with the Software Directive, provides additional arguments in favor of the interpretation of the Directive suggesting that the publication of source code is generally allowed under article 6. Or, at least, this suggests that the French legislators agree with the interpretation suggested in this paper. Let me also mention that some scholars share my impression that the Loi DADVSI assumes the existence of a kind of "right to publish one's source code". For instance, by Prof. Lucas,[278] who – at the same time – stressed that this "right" is limited in the Loi DADVSI by the need of avoiding damages to effectiveness of the DRM system. That having been said, the fact that this limitation is stated as a kind of exception to the right to publish the source code[279] and the fact that the burden of proof about the inappropriateness of the publication is on the DRM controller seem to confirm that such a right indeed exists. Moreover, the narrow limitation to the freedom of publishing source code is – as I already recalled – limited to the case where the information necessary for interoperability has been disclosed by the controller of the platform. That's why, *a fortiori*, there must be (at least in France, but – I would argue – in any country implementing the Software Directive in a way which is coherent with its *ratio*)

[277] Id., pp. 111-113.

[278] I am grateful to Andrè Lucas (Université de Nantes-Cedex) for his presentation *Mesures techniques de protection et d'information et interopérabilité dans le code de la propriété intellectuelle française* at the SIAE seminar "Nuove prospettive dell'accesso ai contenuti e della interoperabilità nell'evoluzione del diritto d'autore contemporaneo" (10th of October 2007, Milan) and for the following discussion.

[279] See also GAUBIAC, *Interopérabilité et Droit de Propriété Intellectuelle*, pp. 111-113.

a right to publish the source code in cases in which one finds the information himself/herself and there is no security risk in publishing it.[280]

7. If uncertainty is significant only for open source projects, should we really care?

According to my thesis, the European Software Directive poses some obstacles to open source or other distributed decompilation projects. Also the favor of US courts for clean-room reverse engineering is mainly problematic for open source decompilation projects. However, one could argue, I did not demonstrate that these obstacles are major and make software reverse engineering impossible. Moreover, I did not show that they are a major concern for commercial software houses. Even though I tried to demonstrate that these obstacles are not coherent with the general philosophy of intellectual property protection of software, and I also tried to show that eliminating them would not have major negative effect on incentives to innovate, one could still legitimately ask himself, "should I really care about these details about intellectual property in a very specific field?". I argue that we should care, and quite a lot, about these obstacles to the obtainment of horizontal interoperability by free software and/or open source (FLOSS) projects, because they are probably the only, and surely the most credible, threats to established commercial incumbent in the field of operating systems and similar software platforms.

In software markets, there are only a few undertakings that may have an incentive to realize a substitute for a very successful and established product, unless they are able to provide an extremely improved piece of software (something that is unlikely, in a field characterized by incremental innovation). An exception (as I will discuss in the third paper) are software platforms wanting to mitigate the market power of some producers of complements (or simply wanting to leverage their power in adjacent markets) and having the possibility of leveraging market power to generate network effects through bundling. However, holding some market power is a prerequisite of this strategy. Thus, and even though one may hope that dominant incumbents in different markets will try to threat dominant firms in other markets, a scenario in which Google bets its financial resources to start a war against Microsoft in the field of PC operating systems does not seem particularly likely to me. Especially because the most likely effect of this kind of strategy would be a lowering of the profits of both Google and Microsoft. A tacitly collusive equilibrium, in which market leaders keep and safeguard their own quasi-monopolies appears to be more likely, and it is in doing so that bundling strategies are especially "helpful" (as I will show in the next paper). Luckily for competition (and, I submit, for consumers), another potential exception are FLOSS developers, because they are among the few that may want to pursue a "me too" competition, even in cases in which the chances of the project being successful are low (or maybe precisely because of that, in order to receive reputational recognition). In fact, open source developers do not need to break even on costs from the financial/accounting point of view. They may be perfectly happy with a project not making any money, but representing a viable alternative to an existing dominant product (even if, maybe, just for a limited share of quite sophisticated users liking to install their own software and having a special awareness of existing alternative to the most widespread products). Indeed, there is a high likelihood that a functional clone of an existing product will never run a significant profit and a similarly high likelihood that horizontal interoperability will never be perfect (so that significant indirect network effects will always protect products like Windows). Hence, the effect of successful open source projects may just be that their developers improve their reputation (and, maybe, get hired by some commercial software house, possibly even Microsoft), while – at the same time – dominant players will remain dominant, but will have some more difficulties in exercising their market power. But such an outcome may be quite positive, from the point of view of social welfare.

The relevance of open source as a competitive threat to dominant player in several software markets that were almost completely uncontested until some years ago did not escape the attention of other commercial software houses. So Sun Microsystems joined with the open source community to develop a real competitor of Microsoft Office: Open Office, which was based of the "opened" source code of Sun's project Star Office, which – as a commercial product – looked highly unlikely to represent a competitive threat for Microsoft. Similarly, several commercial software house, like Google[281] (but also IBM or Intel), help the open source community in various ways, and surely in their own interest. Indeed, FLOSS seems to be increasingly

[280] A point deserving a comment is the fact that the French legislator – despite its attention to the theme of interoperability – excluded software from the "contents" to which the interoperability provisions of the Loi DADVSI apply. Apparently, that has been done because the French legislator considered that the issue of software interoperability was already regulated by the Software Directive (and its French implementation).

[281] For instance, http://mailman.fsfeurope.org/pipermail/press-release/2008q1/000201.html.

perceived as a powerful tool to compete against dominant incumbent in markets characterized by very high network effects[282]. And this perception seems to be shared also by the software house that is considered – and probably considers itself – the very nemesis of the FLOSS software movement, Microsoft, the CEO of which asserted that:

> We've been competing against free alternatives for years; Star Office to Open Office, da-deet, da-deet, da-deet. I'm not saying it's not a real competition. Maybe the world has exactly what it wants. It has us moving fast and hard, keeping our prices down. And even if the other guy doesn't get any traction or momentum, the other guy has no cost structure. So they are not going away either. Maybe that is what the world wants. Put the courts aside, we have a lot of competition. We have to outrun this phenomenon. And I think we're doing a good job of it. But if we let up for a minute, I think there are issues. You would say to me, if we stop, if we're no longer more functional, if we're no longer a time saver, if we're no longer compatible, then that other stuff will gain traction.[283]

Hence, open source does represent a competitive threat for commercial software houses, and in particular for the dominant ones, which no other traditional commercial developer would likely threaten, simply because an years-long "competitive siege" would be needed, and commercial software houses, ad Steve Ballmer puts it do have a "cost structure" and frequently binding financial constraints.

Another recent hint at the relevance of the open source threat for Microsoft – if needed – are the significant efforts of the software house in order to have its MS-OOXML format for office suites documents approved as an ISO standard, after that the open source supported ODF format received the same approval and was hence officially supported by several national and international institutions.[284] In an official document addressed to the US Security and Exchange Commission, Microsoft recently confirmed – in a less colorful way – the concepts expressed by its CEP, starting its description of the main risk factor it faces with the following words:

> **Challenges to our business model may reduce our revenues and operating margins.** Our business model has been based upon customers paying a fee to license software that we develop and distribute. Under this license-based software model, software developers bear the costs of converting original ideas into software products through investments in research and development, offsetting these costs with the revenue received from the distribution of their products. Certain 'open source' software business models challenge our license-based software model. […] A number of commercial firms compete with us using an open source business model by modifying and then distributing open source software to end users at nominal cost and earning revenue on complementary services and products. These firms do not bear the full costs of research and development for the software. Some of these firms may build upon Microsoft ideas that we provide to them free or at low royalties in connection with our interoperability initiatives. To the extent open source software gains increasing market acceptance, our sales, revenue and operating margins may decline.[285] […]
>
> Open source software vendors are devoting considerable efforts to developing software that mimics the features and functionality of our products, in some cases on the basis of technical specifications for Microsoft technologies that we make available. In response to competition, we are developing versions of our products with basic functionality that are sold at lower prices than the standard versions. These competitive pressures may result in decreased sales volumes, price reductions, and/or increased operating costs, such as for marketing and sales incentives, resulting in lower revenue, gross margins and operating income.[286]

The aforementioned quotation comes from the heading of the section "Risk Factors" in Microsoft's report to SEC. Hence it is reasonable to suggest that Microsoft – the most powerful incumbent in the consumer

---

[282] Both direct and indirect, but the latter case seems to be the more relevant (OpenOffice – Linux). An interesting example is provided by Nokia, which recently bought the smart-phone operating system Symbian (*see*, among many, *Nokia to Open Access to Mobile Software*, by Kevin O'Brien, June 25, 2008 http://www.nytimes.com/2008/06/25/technology/25nokia.html), precisely in order to open it and compete against Microsoft and possibly Apple, which is adopting an opposite and very "close" strategy… *See also* EVANS, et al., *Invisible Engines*.

[283] Steve Ballmer interviewed by Forbes.com. Daniel Lyons, *Ballmer, Bemused*, Forbes.com – Computer Hardware & Software, March 23, 2006, available at http://www.forbes.com/2006/03/22/ballmer-microsoft-linux-cz_df_0322microsoft.html (last visited July 21, 2008).

[284] See http://en.wikipedia.org/wiki/Office_Open_XML.

[285] Microsoft Corporation, Annual Report to SEC on Form 10-K, Commission file number 0-14278 (hereinafter, Microsoft Report to SEC), page 12, heading of *Item 1A. Risk Factors*. (Available at http://www.sec.gov/Archives/edgar/data/789019/000119312508162768/d10k.htm; last visited August 4, 2008.)

[286] Microsoft Report to SEC, p. 13.

software industry, arguably extending its market power to low-end servers[287] – considers that the open source model of software development, in combination with the investments of potential or actual commercial competitors, is the most credible threat to its quasi-monopolistic competition in several markets (in particular, one may think about personal computer operating systems and office suites). Moreover, Microsoft argues that – even in the absence of an appreciable decrease in its market shares – this competitive threat prevents the software house from fully exploiting profit opportunities in these markets and forces the firm to make available low cost versions of its products. Obviously Microsoft also suggests that this competition may decrease incentives to innovate and not only Microsoft's profits. However, no evidence to support this possibility is provided and the self-interest of Microsoft in arguing in this sense is plainly evident. In fact, nothing in the report suggest that Microsoft is not able to recoup its research costs or that research and development investments had to be reduced, despite various statements suggesting that this may be the case (which are normal in a report that is supposed to show possible risks for stake holders). Actually, in the trend of Microsoft's investments, one can find no evidence that investments in R&D are severely adversely affected by competition:

> During fiscal years 2008, 2007, and 2006, research and development expense was $8.2 billion, $7.1 billion, and $6.6 billion, respectively. These amounts represented 14%, 14%, and 15%, respectively, of revenue in each of those years. We plan to continue to make significant investments in a broad range of research and product development efforts.[288]

8. Conclusions

The detailed limits and conditions necessary to enjoy the decompilation exception of article 6 of the Software Directive; the favor of US courts for clean-room reverse engineering and their caution in applying the fair use test the decompilation cases; the purpose-bound decompilation exception of the DMCA; all these legal norms share a significant level of prudence, in order to prevent possible future risks. However, as Spoor puts it, "the reverse engineering provisions in the Directive [as, I would add, all the aforementioned norms] can probably be characterized as a rare instance of 'legislation ahead of its time'."[289] The problem is that "[i]t must be doubted whether that time will ever come."[290] Legislating in this way in the high-tech software industry is not wise: being prudent could simply mean being irresponsible, if this prudence goes in the direction of limiting one of the few tools that could be used to (try to) compete with powerful dominant incumbents, reinforcing their dominant positions. Notice that I am not simply forgetting the fact that reverse engineering could be used also "against" other – non-dominant – competitors. Indeed, decompilation is so costly and time consuming that it could be reasonably used only when issues at stake are huge, when the competition is for the control of a market, where an established incumbent looks unchallengeable. When competitors are on a substantially plain competitive field, then it is cheaper for each of them to develop software from scratch, instead of starting from the decompilation of their opponent's programs. Technologists confirm that "[i]t is probably easier and almost certainly much more attractive for competitors to develop completely new software, rather than to reverse engineer existing code."[291] Hence, it is only when something else is at stake that decompilation is normally performed. And this "something else" is typically represented by the possibility of accessing the "rent" represented by very significant indirect and direct network effects, created by the existence of third parties' compatible applications and user created data (the value of which grows with the market share of the incumbent software house, and the existence of which increases the incumbent's market power).

When the Software Directive was adopted, several commentators[292] pretended that the European legislators were being too bold in favoring interoperability. Actually, article 6 of the Software Directive was needed, in civil law countries, to reach the same results that – in the US – was possible to reach through jurisprudential

---

[287] See *Microsoft IV* Decision (*supra* note 228).

[288] Microsoft Report to SEC, p.8

[289] Ibidem.

[290] Ibidem.

[291] See JOHNSON-LAIRD, *Software Reverse Engineering*, , but see also SPOOR, *Copyright Protection and Reverse Engineering*, p. 1078.

[292] See, in particular, the debate hosted by the European Intellectual Property Review in 1989-1991. CORNISH, *Inter-operable Systems*, and WILLIAM T. LAKE, *Seeking Compatibility or Avoiding Development Costs? A Reply on Software Copyright in the EC*, 11 European Intellectual Property Review, 431--434 (1989), but also CAROLINE MEYER & MICHEL COLOMBE, *Seeking Interoperability: An Industry Response*, 12 European Intellectual Property Review, 79--83 (1990); CAROLINE MEYER & MICHEL COLOMBE, *Interoperability still Threatened by EC Software Directive: A Status Report*, 12 European Intellectual Property Review, 325--329 (1990); ROBERT J. HART, *Interfaces, Interoperability and Maintenance*, 13 European Intellectual Property Review, 111--116 (1991).

interpretation (and this need was present also in the UK, where the fair dealing exceptions are far from being as flexible as US fair use). However, the Directive implemented the narrowest possible decompilation exception. And did that myopically thinking only of the two interest groups, which were facing each other in Brussels at the time, both consisting of commercial software houses, even if of different dimensions. It is just because of the civil law nature of the majority of European legal systems that, in Europe, support for reverse engineering and interoperability is more explicit than in the US. European legal systems just needed a higher level of codification, but these activities are likely not much easier or safer in Europe than in the US. Quite to the contrary, several kinds of decompilation projects may arguably be performed under the fair use doctrine, but not under article 6 of the Software Directive. Maybe, the degree of legal uncertainty concerning reverse engineering to achieve interoperability is slightly lower in Europe than in the US, but in both countries commercial software house have nothing to fear, as long as they do not infringe copyright re-implementing discovered interfaces.

What is more important, on both sides of the Atlantic, open source developers – likely the only (or, at least, the most) credible competitive threat to established incumbents in several software markets, and in particular in that of operating systems – face particular difficulties. These obstacles arise from the fact that reverse engineering and interoperability exceptions have been established in the narrowest possible way, obviously thinking about traditional commercial software houses. The reason for these "exceptions" being so narrow is that US Courts were careful in not setting excessively wide precedents (and – after all – they were just facing cases concerning traditional software houses). Similarly, the European legislators (when drafting the Directive) were faced by the lobbying of big US commercial software houses confronted by the lobbying of smaller (but still traditional) European software houses. Indeed, a balance was struck between the positions of these undertakings, but likely did not take into consideration enough theoretical and technical reasoning, that was already present and quite clear in the literature,[293] but not yet represented by a significant opinion movement.

As this paper showed, software reverse engineering is completely free only in the form of black box analysis,[294] the results of which may be freely disclosed.[295] It also showed that decompilation in order to obtain interoperability is allowed both in the EU and in the US, but it argued that several specific qualifications with respect to the freedom to decompile are not strictly necessary and – what is more worrying – may have some especially harmful unintended consequences, in particular hindering open source decompilation projects. Furthermore, it showed that a clear-cut safe harbor would likely not create major costs in terms of reduced incentives to innovate and would go in the direction of making feasible big reverse engineering projects performed in a decentralized way, as in the context of an open source community.

Among the reasons in favor of a generic safe harbor there are: technical reasons (in many cases, you don't really know what you are decompiling, until you actually start to analyze a software, typically making intermediate copies and derivative works, and hence potentially violating copyright); difficulties of monitoring and enforcement (in a traditional firm, a decompilation project may be kept secret, especially if the goal is not to realize an interoperable software program, but just to learn other kinds of information); actual violations of copyright – apart from the literal violation concerning intermediate copies for "studying purposes" during decompilation – arise during the reimplementation phase (and since evidence for "illegal" decompilation should frequently be looked for in code that is actually released, there are even less advantages in discussing the legality of decompilation in itself).

In the EU, purpose-bound decompilation (i.e. decompilation allowed only to attain interoperability) prohibits some kinds of interoperability for legitimate purposes (e.g. for error detection, to investigate suspect copyright violations) and – more generally – impedes an economically healthy means to discover unprotected ideas, resulting in an unprecedented reinforcement of trade secret. Fortunately, since interoperability remains the main goal of economically relevant decompilation projects, article 6 of the Software Directive seemed to allow firms to perform the crucial form of decompilation; in fact, the second paper also showed that –

---

[293] *See* CORNISH, *Inter-operable Systems*.

[294] *See* GUGLIELMETTI, *L'invenzione di software (2nd ed.)*, p. 256 (fn 116), where the author argues that "black box analysis" is allowed, as long as the decompilation project is performed by a subject legitimately owning a copy of the analysed piece of software.

[295] In fact, I cannot refrain from noting that the decision of the European legislator to explicitly allow black box reverse engineering at article 5 of the Software Directive is somehow worrying: in fact, it is a hint at how detailed and pervasive civil law needs to be in order to allow common sense exceptions to copyright, the absence of which would likely violate constitutional rights. But it is possible to argue also that this exception has been introduced just for the sake of legal clarity and certainty.

despite some different opinions in the literature – the literal text of the Directive and both legislative history and recent Decisions of the Commission (confirmed by the European Courts) make clear that the interoperability exception concerns both vertical and horizontal interoperability (i.e. it is legitimate to decompile a piece of software to realize another program that is compatible with third party software capable to interoperate with the decompiled one). However, the paper also showed that (and again in the EU), limitations to the disclosure of the information obtained through reverse engineering put the open source model of software development at a disadvantage, with respect to the integrated and closed proprietary model (which was likely the only one envisaged at the time of the Software Directive drafting).

Finally, the paper showed that the US legal setting is more flexible and capable of accommodating more kinds of legitimate and pro-competitive decompilation efforts; nonetheless, also in the US the open source model of software development may be somehow disadvantaged (with respect to commercial software houses) in so far as the clean-room process of reverse engineering is perceived (by some courts and – in particular – by people operating in the software industry) as a necessary condition for legitimate reverse engineering. Hence, a clear-cut legislatively created safe harbor for software reverse engineering would be useful in the United States as well. To be sure, in principle, it would be sufficient to consider the clean-room process not as an actual requirement for legitimate decompilation, but as a mental experiment showing which part of the results of a decompilation projects could be used in the re-implementation phase. However, not only would legal certainty be enhanced by a clearer explicit exception, but also the "expressive value of the law"[296] would play in the direction of an interoperability-friendly environment.

In 1994 Spoor concluded a paper[297] about reverse engineering under the EC Software Directive arguing that "[w]hile the EC Directive can be said to have struck a fair balance between the main interests at stake, it may at the same time leave too little room for other interests, as well as for new developments."[298] The analysis provided by the paper at hand confirms that early impression and could also be considered a "case study" concerning the effect of drafting rules in the field of technology under the pressure of various kinds of lobbies. Indeed, the reason for which Article 6 of the software directive, as it is written today, does not make much sense from an economic point of view, is that it is simply the midway solution between two lobbying groups: the EICS on one side and SAGE on the other. At the time of drafting, the European legislator essentially decided to side with the EICS, but also inserted a number of limitations to the freedom to decompile, since these limitations were politically useful to pay lip service to the interest of SAGE and – given the working of the software industry at the time – they did not pose major obstacles to the typical market strategies adopted by firms like the traditional software houses forming the ECIS. All amendments trying to set a clear-cut rule in favor of decompilation or restricting it in a clear and very significant way have been rejected[299]. Ambiguity has not been an unintended effect, quite the opposite, it has been actively used as a tool for finding consensus or avoiding a direct clash with this or that interest group. As a commentator puts it, "much of article 6's wording is based on compromises which do not solve the conflicts of interests but rather leave them to the European Court of Justice to solve."[300] Unfortunately, Europe remained relatively marginal in software litigations, so that the European Courts did not clarify much of the interpretations of article 6.[301] The result has been a rule that – after two decades – clearly shows its age and it is less and less able to accommodate new models of software development. More importantly, during these twenty years, precisely what the Commission was trying to avoid happened: a single firm, Microsoft, steadily became the leader of a significant part of the software market, controlling a *de facto* standard and leveraging it from the PC operating systems market to adjacent markets (as found by the Commission and confirmed by the CFI). At the same time, the ambiguous wording of Article 6 gave several commentators[302] the possibility to credibly

---

[296] About the notion of "expressive value of the law," see IAN AYRES, *Menus Matter*, 73 The University of Chicago Law Review, (2006) and Lectio Magistralis of Stresa Lectures Series, 10th of July 2006 ("A Theory of Default and Altering Rules in Contract Law").

[297] SPOOR, *Copyright Protection and Reverse Engineering.*

[298] Id., p. 1085.

[299] *See* GUGLIELMETTI, *in* Analisi e decompilazione.

[300] SPOOR, *Copyright Protection and Reverse Engineering*, 1070, attributing the paternity of this point of view to Schulte (D. Schulte, *Der Referententwurf eines Zweiten Gesetzes zur Aenderung des Urheberrechtsgesetzes. Ausgewahlte Auslegungsfragen der EG-Richtlinie uber den Rechtsschutz von Computerprogrammen*, 8 Computer und Recht 648, 653, 1992).

[301] A partial exception may be the fact that the Court of First Instance sided with the European Commission in its general (and relatively wide) interpretation of the concept of interoperability in the context of the European Microsoft case, but this was a competition policy case, not an intellectual property one. See the first and third papers of this dissertation for some more details.

[302] For a paradigmatic example, *see* HART, *Interoperability Information.*

argue that interoperability was not a clear major goal of the Software Directive or, at least, that this goal had to be qualified in several ways (just vertical, not horizontal; just partial, not as plug-in replaceability; etc.).

Today, with insights, it is quite clear that software reverse engineering (and decompilation in particular) is not a convenient tool for late comers wanting to "appropriate" the significant investments bared by first comers in software markets. It is also clear that technology does protect in a more than effective way the secrets hidden in the compiled binary code distributed by commercial software houses. Hence, there are no reasons to legally increase this technical protection. The need to formally violate copyright law in order to decompile a software derives from the legal fiction of considering also object code (and not just source code) as an object of copyright protection. This fiction was needed in order to prevent software piracy and it is appropriate, but the existence of this fiction does not make decompilation an "improper means" to discover a trade secret. In civil law countries an exception to clearly state that is probably necessary. That is fine. Nothing more is needed and general copyright principles are sufficient to prevent abuses of this freedom. Other limitation are just unnecessary and are likely to have – soon or later – unintended negative impacts on innovation.

Both in the EU and in the US, a clear-cut safe harbor for software decompilation would have no significant economic side-effects, would increase legal certainty and could foster competition precisely against the most dominant of software incumbents.

Bibliography

JOHN ABBOT, *Reverse Engineering of Software: Copyright and Interoperability*, 14 J.L. & Inf. Sci., 7 (2003)
JEFFREY A. ANDREWS, *Comment, Reversing Copyright Misuse: Enforcing Contractual Prohibitions of Software Reverse Engineering*, 41 Hous. L. Rev., 975 (2005)
IAN AYRES, *Menus Matter*, 73 The University of Chicago Law Review, (2006)
JONATHAN B. BAKER, *Beyond Schumpeter vs. Arrow: How Antitrust Fosters Innovation*, (June, 2007)
JONATHAN BAND & MASANOBU KATOH, *Interfaces on Trial -- Intellectual Property and Interoperability in the Global Software Industy*, (Jonathan Band ed., Westview Press First ed, Boulder, Colorado. 1995)
JAMES BESSEN & ROBERT M. HUNT, *An Empirical Look at Software Patents*, Federal Reserve Bank of Philadelphia Working Papers 03-17/R (March, 2004)
JAMES BESSEN & ROBERT M. HUNT, *An Empirical Look at Software Patents*, 16 Journal of Economics and Management Strategy, 157--189 (2007)
JAMES BESSEN & MICHAEL J. MEURER, *Patent Failure*, (James Bessen ed., Princeton University Press. 2008)
FABRIZIO BROCK, *La disciplina del 'reverse engineering' nella legge di attuazione della Direttiva CEE sul software*, 1 Rivista di Diritto Industriale, 267--279 (1993)
J. BUCHANAN & Y. YOON, *Symmetric Tragedies: Commons and Anticommons Property*, 43 Journal of Law and Economics, 1-13 (2000)
DANIELA CATERINO, *Software e rifiuto di licenza del codice sorgente*, Annali Italiani di Diritto d'Autore, 388 (2004)
W. R. CORNISH, *Inter-operable Systems and Copyright*, 11 European Intellectual Property Review, 391--393 (1989)
B. CZARNOTA & R. J. HART, *Legal Protection of Computer Programs in Europe: A Guide to the EC Directive*, (Butterworths Tolley. 1991)
GIUSEPPE DARI-MATTIACCI & FRANCESCO PARISI, *Substituting Complements*, 2 Journal of Competition Law and Economics, 333--347 (2006)
JAMES L. DAVIS, *Is Interoperability just for Those Who Can Hack It? The Application of the DMCA Interoperability Exceptions in the Consumer Electronics Industry*, 2005 University of Illinois Journal of Law, Technology, and Policy, 141 (2005)
BEN DEPOORTER & FRANCESCO PARISI, *The Market for Intellectual Property: The Case of Complementary Oligopoly*, *in* The Economics of Copyright: Developments in Research and Analysis, (W. Gordon & R. Watt eds., 2003)
ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 2*, 22 European Intellectual Property Review, 56-68 (2000)
ESTELLE DERCLAYE, *Software Copyright Protection: Can Europe Learn from American Case Law? -- Part 1*, 22 European Intellectual Property Review, 7-16 (2000)
LOTHAR DETERMANN, *Dangerous Liaisons -- Software Combinations As Derivative Works? Distribution, Installation, And Execution Of Linked Programs Under Copyright Law, Commercial Licenses, And The Gpl*, 21 Berkeley Technology Law Journal, 1421 (2006)
DAVID S. EVANS, et al., *Invisible Engines -- How Software Platforms Drive Innovation and Transform Industries*, (David S. Evans ed., MIT Press First paperback ed. 2008)
JOSEPH FARRELL & PHILIP J. WEISER, *Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age*, 17 Harvard Journal of Law & Technology, 85 (2003)
PAOLA A. E. FRASSI, *Innovazione derivata, brevetto dipendente e licenza obbligatoria*, I Rivista di Diritto Industriale, 212--226 (2006)
JOSHUA S. GANS, et al., *The Impact Of Uncertain Intellectual Property Rights On The Market For Ideas: Evidence From Patent Grant Delays*, NBER Working Paper 13234 (July, 2007)
YVES GAUBIAC, *Interopérabilité et Droit de Propriété Intellectuelle (with en. translation: Interoperability in Intellectual Property Law)*, 211 Revu Internationale du Droit d'Auteur, 91--139 (2007)
GUSTAVO GHIDINI, *Profili evolutivi del diritto industriale. Proprietà intellettuale e concorrenza*, (Giuffrè, Milano. 2001)
GUSTAVO GHIDINI, *Intellectual Property and Competition Law. The Innovation Nexus*, (Edward Elgar. 2006)
GUSTAVO GHIDINI & VALERIA FALCE, *Recent developments in Italian regulation of trade and industrial secrets: A patent contradiction of the patent regime?*, paper presented at the 3rd Annual Workshop on the Law and Economics of Intellectual Property and Information Technology, 5-6 July, 2007, Queen Mary, University of London (July, 2007)

GIANVITO GIANNELLI, *in* Commentario Breve alle Leggi su Proprietà Intellettuale e Concorrenza, (Luigi Carlo Ubertazzi ed., 2007)
WENDY J. GORDON, *Assertive Modesty: An Economics of Intangibles*, 94 Colum. L. Rev., 2579--2593 (1994)
GIOVANNI GUGLIELMETTI, *Le topografie dei semiconduttori*, AIDA, 191 (1992)
GIOVANNI GUGLIELMETTI, *Analisi e decompilazione dei programmi*, *in* La legge sul software, 152--201 (Luigi Carlo Ubertazzi ed., 1994)
GIOVANNI GUGLIELMETTI, *L'invenzione di software -- brevetto e diritto d'autore*, (Giuffrè second ed, Milano. 1997)
GIOVANNI GUGLIELMETTI, *La proposta di Direttiva sulla brevettazione delle invenzioni in materia di software*, 1 Rivista di diritto industriale, 438--463 (2002)
ANDREI HAGIU, *Two-sided Platforms: Pricing and Social Efficiency*, Harvard Business School and Research Institute of Economy Trade and Industry working paper, Cambridge, Mass. (2005)
ANDREI HAGIU, *Pricing and Commitment by Two-Sided Platforms*, 37 Rand Journal of Economics, 720--737 (2006)
ROBERT J. HART, *Interfaces, Interoperability and Maintenance*, 13 European Intellectual Property Review, 111--116 (1991)
R. J. HART, *Interoperability Information and the Microsoft Decision*, 28 European Intellectual Property Review, 361--365 (2006)
MICHAEL A. HELLER, *The Tragedy of the Anticommons: Property in the Transition from Marx to Markets*, 111 Harvard Law Review, 621--687 (1998)
MICHAEL A. HELLER & REBECCA S. EISENBERG, *Can Patents Deter Innovation? The Anticommons in Biomedical Research*, 280 Science, 698--701 (1998)
A. JOHNSON-LAIRD, *Software Reverse Engineering in the Real World*, 19 University of Dayton Law Review, 843 (1994)
DENNIS S. KARJALA, *Copyright Protection of Computer Documents, Reverse Engineering, and Professor Miller*, 19 University of Dayton Law Review, 975 (1994)
WILLIAM T. LAKE, *Seeking Compatibility or Avoiding Development Costs? A Reply on Software Copyright in the EC*, 11 European Intellectual Property Review, 431--434 (1989)
JOSH LERNER & JEAN TIROLE, *Some Simple Economics of Open Source*, 50 The Journal of Industrial Economics, 197--234 (2002)
FRANÇOIS LÉVÊQUE, *Innovation, Leveraging and Essential Facilities: Interoperability Licensing in the EU Microsoft Case*, 28 World Competition, 71--91 (2005)
JACQUELINE LIPTON, *The Law of Unintended Consequences: The Digital Millennium Copyright Act and Interoperability*, 62 Washington and Lee Law Review, 487 (2005)
STEPHEN M. MAURER & SUZANNE SCOTCHMER, *Open Source Software: The New Intellectual Property Paradigm*, NBER Working Paper No. 12148 (March, 2006)
C. R. MCMANIS, *Intellectual Property Protection and Reverse Engineering of Computer Programs in the United States and the European Community*, 8 Berkeley Technology Law Journal, 25 (1993)
CAROLINE MEYER & MICHEL COLOMBE, *Seeking Interoperability: An Industry Response*, 12 European Intellectual Property Review, 79--83 (1990)
CAROLINE MEYER & MICHEL COLOMBE, *Interoperability still Threatened by EC Software Directive: A Status Report*, 12 European Intellectual Property Review, 325--329 (1990)
ANDREA OTTOLIA & DAN WIELSCH, *Mapping the Information Environment: Legal Aspects of Modularization and Digitalization*, 6 Yale Journal of Law and Technology, 174 (2004)
FRANCESCO PARISI, et al., *Simultaneous and Sequential Anticommons*, 17 European Journal of Law and Economics, 175--190 (2004)
J. H. REICHMAN, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Columbia Law Review, 2432 (1994)
MARCO RICOLFI, *Is There an Antitrust Antidote Against IP Overprotection within Trips?*, 10 Marq. Intell. Prop. L. Rev., 305--367 (2006)
PAMULE SAMUELSON & S. SCOTCHMER, *The Law and Economics of Reverse Engineering*, 111 Yale Law Journal, 1575--1663 (2002)
PAMELA SAMUELSON, et al., *A Manifesto Concerning the Legal Protection of Computer Program*, 94 Columbia Law Review, 2308--2431 (1994)
JOSHUA SCHWARTZ, *Thinking Outside the Pandora's Box: Why the DMCA Is Unconstitutional under Article I, § 8 of the U.S. Constitution*, 10 Journal of Technology Law and Policy, 93 (2005)
DEANA SOBEL, *A Bite out of Apple? iTunes, Interoperability, and France's Dadvsi Law*, 22 Berkeley Technology Law Journal, 267 (2007)

SEUNGWOO SON, *Can Black Dot (Shrinkwrap) Licenses Override Federal Reverse Engineering Rights?: The Relationship Between Copyright, Contract, and Antitrust Laws*, 6 Tulane Journal of Technology and Intellectual Property, 63 (2004)
JAAP H. SPOOR, *Copyright Protection and Reverse Engineering of Software: Implementation and Effects of the EC Directive*, 19 U. Dayton L. Rev., 1063 (1994)
MIKKO VÄLIMÄKI, *Software Interoperability and Intellectual Property Policy in Europe*, 3 European Review of Political Technologies, 1--11 (2005)

SOFTWARE INTEROPERABILITY AND MODULARITY
Competition Policy and the Complementarity between Tying and Information-Withholding

Third paper of the dissertation project:
Software Interoperability: Issues at the Intersection between Intellectual Property and Competition Policy

Federico Morando
(federico.morando@email.it)

Ph.D. Programme in Comparative Analysis of Law, Economics and Institutions

October 13, 2009

The Interuniversity Centre for the Comparative Analysis of Law and Economics, Economics of Law, Economics of Institutions

SOFTWARE INTEROPERABILITY AND MODULARITY
Competition Policy and the Complementarity between Tying and Information-Withholding

ABSTRACT

This paper claims that, in software markets, information-withholding (or "refusal-to-deal") strategies are normally in a complementary relationship with tying (or "predatory-innovation") strategies. Moreover, I maintain that this complementarity is so relevant that dominant platform controllers need to couple both kinds of conduct in order to create significant anti-competitive effects. Hence, the paper argues that – in order to safeguard competition – mandatory unbundling (i.e. mandating a certain degree of modularity in software development) could be an appropriate (and sometimes even preferable) alternative to the mandatory disclosure of interoperability information.

However, an objection to the implementation of mandatory modularity (unbundling) may be that competition authorities should determine a minimum price for the unbundled product (and that this exercise may be especially complex). As I will show in the last part of the paper, this is not necessarily true. Indeed, in software markets, mandatory unbundling (modularity) may be a useful policy even if the only constraint on the price of the unbundled good is that of non-negativity, as I will demonstrate applying suggestions coming from the literature concerning complementary oligopoly (in particular, modifying models proposed by Dari-Mattiacci, Depoorter and Parisi[1]).

The recent European competition policy case involving Microsoft offers interesting examples of both information withholding and tying. The case in question is extensively analyzed and commented on through the paper, with references both to the European Commission's Decision and to the recent ruling of the Court of First Instance.

[1] See BEN DEPOORTER & FRANCESCO PARISI, *The Market for Intellectual Property: The Case of Complementary Oligopoly*, *in* The Economics of Copyright: Developments in Research and Analysis (W. Gordon & R. Watt eds., 2003) and GIUSEPPE DARI-MATTIACCI & FRANCESCO PARISI, *Substituting Complements*, 2 Journal of Competition Law and Economics, 333—347 (2006).

PAPER 3 – TABLE OF CONTENTS

1. Link with the first and second papers of the dissertation

Interpreting the law as suggested in the first two papers of this dissertation – and possibly modifying it in a limited way, as I proposed – would reduce the likelihood of market failures in software markets. That would be done by reducing uncertainty concerning both the legal status of interoperability information and software reverse engineering. The interpretation of copyright law I recommended would decrease legal uncertainty and the fear of legal actions from big players. Competition could be somewhat increased, but not so much as to create market failures due to excessively easy appropriability (as long as the cost of reverse engineering and the advantage of being first to the market remain as significant as they are today). In this setting, a super-dominant incumbent would be much more likely than today to loose its position just in case of prolonged periods of reduced innovation, because the barrier to entry represented by the existence of significant indirect network effects would be progressively lowered.[2] Notice that, in this legal setting, the so-called "application barrier to entry" (protecting software platform incumbents) would still exist for technical reasons: it would just not be strengthened too much by the law.[3] In other words, in the first two papers of this dissertation I already addressed competition policy concerns, but from an *ex ante* perspective, tailoring intellectual property in a pro-competitive way.[4] Under this approach, crucial incentives to innovate may be preserved, but super-dominant positions are unlikely to be established and to become too stable simply by chance: catching-up from competitors is frequently possible, at least if first comers slow down the pace of innovation. At the same time, if they keep innovating, dominant firms will likely remain market leaders, but this is not very worrying for consumers: the goal of IP and competition policy is to maintain a significant rate of innovation, not to displace powerful incumbents.[5]

However, no design of intellectual property could eliminate the risk of abuses. It is very well know – both for empirical and theoretical reasons – that the creation of persistent dominant positions in software markets is likely to happen in any case and that these positions could be abused in several ways, in particular leveraging them in some adjacent profitable markets. This is why the paper at hand adopts the *ex post* perspective of competition policy[6] and tries to focus on some measures capable of further reducing the risk of market failures due to the excessive stabilization of market power in the hand of a few undertakings. More specifically, I think that the *ex post* perspective of competition policy should be qualified. Indeed, antitrust intervention may have at least two goals: (1) preventing the lessening of competition resulting in market failures; (2) restoring competition after these failures or other abuses, through structural or behavioral remedies (this goal is complementary with the first one and it is especially useful to address unpredicted situations). This third paper of my dissertation focuses on the first of these goals, devising a structure of incentives capable of having also *ex ante* effects, *id est* capable of shaping software products and firm strategies in a pro-competitive direction. In particular, the measures, which are able to reach the first goal, are the ones that can influence a given market, without a continuous need for intervention and monitoring. Thus, I will

---

[2] In some cases, application realized – for instance – for a given operating system (say Microsoft Windows) could run *telle-quelles* on a competing operating system (say Linux, complemented by Wine: an open source reimplementation of Windows APIs, discussed at length in the second paper of this dissertation). But – even without perfect plug-in replaceability – the fact of having several Windows APIs implemented under Linux would greatly reduce the cost of porting (i.e. appropriately translating) a given software for the second operating system: for instance, some pieces of commercial software have been ported for Linux, but they work only if also Wine is installed, since they rely on it for several APIs (while they directly use Linux's APIs if those provided by Wine are not fully functional and/or the APIs exposed by Linux are for some reasons preferred).

[3] See the *General Introduction* to this dissertation.

[4] See MARCO RICOLFI, *Is There an Antitrust Antidote Against IP Overprotection within Trips?*, 10 Marq. Intell. Prop. L. Rev., 305—367 (2006), 328, where the author discusses some "generalized and *ex ante* measures" to reach pro-competitive goals through IP, including "the option to design the various features of intellectual property rights, such as the access requirement, the scope of protection, or the limitations and exceptions, in such a way to permanently incorporate pro-competitive features."

[5] The literature about the borders between intellectual property and competition policy is extensive. See, in particular, JOSEF DREXL, *IMS Health and Trinko - Antitrust Placebo for Consumers Instead of Sound Economics in Refusal-to-Deal Cases*, 35 International Review of Intellectual Property and Competition Law, 788—808 (2004) and his discussion of the "Theory of Complementarity". "According to this theory, intellectual property law and competition law pursue identical goals. Both fields of law are designed to promote competition and innovation." Under this theory, "the intervention of competition laws apparently has to depend on the effects of a given IP right and its exercise in the market." That also means that one of the two policies is inappropriate, the other may try to compensate, but not without creating some problems or even some paradoxes. In particular, for many copyright lawyers, European competition law can be seen as "a substitute for the lack of harmonization of the requirements of copyrightability". For instance, the famous case "*Magill* constitutes sound copyright law, but problematic competition law".

[6] See RICOLFI, *supra* note 4 at 328.

concentrate my attention on this kind of rules, devising some principles tailored to favor a dynamic competitive environment in the software market, centered on the concepts of interoperability and modularity.

Ideally, clear rules and the menace of fines and other remedies should be sufficient to avoid any need of actual *ex post* intervention. However, one has to be aware that, in several cases, also strictly *ex post* remedies are necessary, either because an undertaking decided to violate rules that had been established trying to reach the first goal, or because a given situation/behavior/market created unexpected market failures. Yet, the goal of this paper, coherently with the first two of the dissertation project, is to describe an innovation favorable environment, not to discuss optimal government interventions and regulations, hence I will only touch briefly these kinds of situations, remanding to the rich literature existing in this field.[7]

1.1. Introduction to the third paper

On the whole, because of the interaction of trade secret and of the technological peculiarities of software and of a suboptimal intellectual property policy, in software markets certain software platforms could gain the status of *de facto* standard and firms controlling them and their development could achieve a super-dominant[8] (i.e. literally quasi-monopolistic) position. Certainly, becoming market leaders always requires impressive acumen, whether technical or business-related (or, more likely, both). What may be worrying from the perspective of social welfare, however, is that competitive strategies in these markets frequently entail tying, bundling and/or refusal to disclose interoperability information. Moreover, these strategies may have the effect of leveraging a monopolistic position to adjacent markets; or – to use a more neutral wording – of boosting the competitive position of dominant platform controllers in these complementary markets. All that is quite well-known and it has been extensively studied.[9] It is, conversely, less recognized that – as this paper claims – refusal-to-deal (or "non-disclosure"/"information-withholding") strategies are normally complementary with tying[10] (or "predatory-innovation") strategies in software markets; moreover, I will show that this complementarity is so relevant that both strategies may need to be coupled in order to have significant anti-competitive effects. For this reason, mandatory unbundling (i.e. mandating a certain degree of modularity in software development) could be an appropriate alternative to mandatory disclosure of interoperability information. Moreover – and despite the fact that any kind of policy intervention in software (or other innovation intensive) markets may have especially severe unintended consequences – having the

---

[7] Microsoft antitrust cases gave rise to a wealth of literature on remedies and dominant software platforms. Regarding the US Microsoft case, see also J. ZITTRAIN, *The Un-Microsoft Un-Remedy: Law Can Prevent the Problem that It Can't Patch Later*, 31 Connecticut Law Review, 1361 (1999). About the European Microsoft case, see also HARRY FIRST, *Strong Spine, Weak Underbelly: The CFI Microsoft Decision*, NYU Law and Economics Research Paper No. 08-17 (April, 2008) and HARRY FIRST, *Netscape is Dead: Remedy Lessons from the Microsoft Litigation*, New York University - School of Law working paper (August, 2008), which also derives some lessons about antitrust remedies from the overall antitrust history of Microsoft.

[8] A signal of which could be a stable market share above 80-90%. See Opinion of Advocate General Fennelly in Joined Cases C-395/96 P & C-396/96 P, Compagnie Maritime Belge and others v Commission [2000] ECR I-1365, at paragraph 137 (which describes a concept of "superdominance" and highlights the "particularly onerous special obligation" affecting an undertaking which enjoys a position of "overwhelming dominance verging on monopoly"). See also Judgment of the Court of 14 November 1996, Tetra Pak International SA v Commission, Case C-333/94 P [1996] ECR I-05951, at paragraphs 28, 29 and 31 ("where Tetra Pak's "quasi-monopolistic" position, its "almost complete domination of the aseptic markets" and "quasi-monopoly" were referred to as relevant factors justifying that Tetra Pak's conduct on a non-dominated market and having effects on that non-dominated market could be found to be abusive"").

[9] See, in particular, M. WHINSTON, *Tying, Foreclosure and Exclusion*, 80 American Economic Review, 857—873 (1990); MICHAEL D. WHINSTON, *Exclusivity and Tying in U.S. v. Microsoft: What We Know, and Don't Know*, 15 The Journal of Economic Perspectives, 63—80 (2001); D. W. CARLTON & M. WALDMAN, *The Strategic Use of Tying to Preserve and Create Market Power in Evolving Industries*, 33 The Rand Journal of Economics, 194—220 (2002); C. AHLBORN, et al., *The Antitrust Economics of Tying: A Farewell to Per Se Illegality*, The Antitrust Bulletin (2003); BARRY NALEBUFF, *Exclusionary Bundling*, 50 Antitrust Bulletin, 321—370 (2005). See also N. GANDAL, et al., *Ain't it 'Suite'? Strategic Bundling in the PC Office Software Market*, mimeo, Columbia University (2004). For a short survey on the literature about tying and further references, see CORMAC O'DEA, *A Look at the State of Knowledge on Bundling*, 20 Student Economic Review, 53—63 (2006)

[10] With "tying strategies" I mean strategies where two goods cannot be purchased independently, de facto becoming a unique good from the point of view of potential buyers. This tying may even be reinforced technologically, so that it becomes impossible (or quite burdensome) to divide the two goods even after the purchase. In this paper I will not talk extensively about bundling strategies, where two separate goods are both sold as a bundle and as separate goods. Despite the the fact that these strategies are interesting (and their potential relevance for competition policy), I consider them already extensively discussed by several authors of which I essentially share the point of view. See, in particular, NALEBUFF, *Exclusionary Bundling*, p. 16 (notice that this paper is available in more than one format: page references I use concerns the version available at the following link: http://www.olin.wustl.edu/cres/research/calendar/files/ExclusionaryBundlingrevisedrefs4.doc).

possibility of choosing between two instruments (mandating modularity or disclosure), instead of only one (mandating disclosure), could reduce these drawbacks. In principle, competition authorities wanting to intervene in these markets could even leave dominant undertakings free to choose their preferred solution between modularity and disclosure. Ideally, that should be done in cases where either of the two policies may prevent severe anticompetitive effects, while a firm active in the market may hopefully understand which approach minimizes distortions with respect to the optimal path of technological progress and allows for the implementation of the firm's preferred business models.

Furthermore, while mandating disclosure may involve establishing a certain price for such a disclosure and/or a strict monitoring of related pricing policies, an advantage of mandating modularity may be that, in this case, competition policy authorities do not need to determine or monitor pricing choices.[11] This may not be immediately evident. Indeed, considering the critiques moved from part of the literature to the Commission's Decision in the recent European Microsoft antitrust case, an objection to the previous argument could be that – also in the case of mandatory unbundling – one should still determine the minimum price for the unbundled product. Otherwise, the critique would continue, the dominant firm could achieve a result that is equivalent to bundling just selling the formerly tied product at zero price. This may seem a sound argument; still it may be argued, and I do argue, that this is not necessarily the case. To prove this claim, the last part of the paper uses a simple leader-follower oligopoly model with complementary goods to demonstrate that this objection is not well grounded and that – in software markets – mandatory unbundling (modularity) may be a useful policy, and indeed a preferable one to mandatory disclosure. All that, even if the only constraint on the price of the unbundled good is the one of non-negativity.

To sum up, I will show not only that mandating unbundling may be an *appropriate* alternative to mandatory disclosure; I will also show that mandating modularity may indeed be a *preferable* alternative to an obligation to disclose interoperability information.

1.1.1. Plan of the paper

The second section of this paper will discuss the fact that – in general – software platform controllers may have significant incentives to foster innovation, more than to limit it. This idea is coherent with both the traditional one-monopoly-profit theory of the Chicago school and with insights coming from recent two-sided markets model, were central platforms may even play the role of regulation authorities of complementary markets.[12] However, the section also highlights cases, in which this idyllic situation does not take place. These situations, in which platform controllers are more likely to act anti-competitively and to the detriment of innovation, are summarized by the exceptions to the paradigm of the internalization of complementary efficiencies (ICE). In fact, the ICE paradigm offers a useful checklist of situations where a quasi-monopolistic platform controller may leverage its market power in adjacent markets.

Section three deals with some fundamental competition policy concepts, including dominance and super-dominance, while stressing the importance of a manageable and predictable test to analyze abuses, especially in markets as complex as that of software. The section also sketches briefly the salient points of the European evolution of antitrust intervention in markets based on intellectual property.

The Fourth section tries to derive some teachings from Microsoft cases on both sides of the Atlantic (with particular attention to the recent European case). In March 2004, the European Commission (in a Decision that I will refer to as the *Commission's Microsoft Decision*),[13] found that Microsoft broke European competition law, leveraging its near monopoly in the market for PC operating systems in two distinct, but complementary markets: the market for workgroup server operating systems and that of media players. Such a leveraging was based on information-withholding (and the associate refusal-to-deal) as far as workgroup servers were concerned and on the tying of Windows Media Player (WMP) to Windows, in order to defeat competition in the media players market. In September 2007, the European Court of First Instance essentially confirmed the

[11] Notice that also a mandated unbundling activity may need intensive monitoring and – in certain cases – such monitoring may be especially burdensome; however, since also the technical quality of a mandatory disclosure would have to be monitored, eliminating the need for strictly controlling also pricing choices may simplify the overall complexity of the monitoring of antitrust decisions.

[12] See § *2.1. Problems are the exception, not the rule* and accompanying footnotes for references.

[13] *Commission's Microsoft Decision*: Commission Decision of 24 March 2004 relating to a proceeding under Article 82 of the EC Treaty, Case COMP/C-3/37.792 Microsoft. (Available at http://ec.europa.eu/competition/antitrust/cases/decisions/37792/en.pdf.)

Commission's Decisions (with a ruling that I will refer to as *Microsoft CFI*).[14] More generally, the section analyses how platform controllers may use technological tying in order to functionally clone competitor's products and establish their own complementary products as *de facto* standards in adjacent markets. Moreover, the role of various categories of (more or less advanced) users and of original equipment manufacturers is analyzed. The last part of this extensive section shows how tying strategies may be complemented by information withholding strategies and how this complementarity is likely to create significant anticompetitive effects.

Section five summarizes the conclusions derived by the previous analysis and spells out a proposal, allowing competition authorities to take into account the complementarity between tying and information withholding in deciding cases where interoperability and modularity are focal issues.

Section six modifies the models of complementary oligopoly provided by Depoorter and Parisi (2003)[15] and Dari-Mattiacci and Parisi (2006)[16] to analyze, in particular, the economic efficiency of the European Commission Decision in its antitrust case against Microsoft. Despite the fact that this analysis applies in particular to the issue concerning the tying of Windows Media Player with Microsoft Windows operating system, several of the provided intuitions apply to the issue of bundling and mandatory modularity in general. The conclusion of this analysis is that the Decision of the European Commission concerning the tying of Windows Media Player, far from being ineffectual – as has sometimes been suggested – allows Microsoft to perform strong price competition – with potential benefits for consumers – but prevents it from performing the majority of abusive practices that the literature and this paper have discussed.

Section seven provides some insights on possible future antitrust cases involving Microsoft and issues similar to the ones discussed in the paper at hand.

Section eight concludes. The analysis confirms that bundling or information-withholding strategies, in isolation, may have some anticompetitive effects, but are likely to have either limited consequences or to be easily detectable by antitrust agencies. In fact, one of the two strategies alone needs to be performed in a very strong way, if it has to have the effect of excluding competitors, and this helps in identifying plainly anticompetitive behavior. Instead, violating at the same time the principles of modularity (with tying) and interoperability (avoiding the disclosure of interoperability information), super-dominant undertakings may likely (and in a more concealing way) extend their market power to adjacent market, monopolizing them and – what is possibly even more important – protecting their own core platform monopoly. Overall, the paper finds several limits in the European Commission's Decision concerning the aforementioned issues as they are declined in the Microsoft case. However, the Decision is not completely incompatible with the recommendations spelled out in the work at hand; in fact, I argue that this paper could offer a general framework to reinterpret the rationale of the Decision as a whole and provide additional legal certainty in similar future cases.

2. The risk of 'throwing the baby out with the bathwater'

Software firms generally suggest that (technological) tying occurs because users demand complete and working systems "out of the box"; additionally, people want their system to be (silently) kept updated to be able to allow them to benefit from new technologies (for instance, being able to play MP3 music or DVX videos as they become available). Moreover, tying strategies would serve two other main (legitimate) aims, which I will rephrase in a unique economic problem: solving coordination games between users or users and content producers (allowing them all to shift to a new, better technology). Hence, government intervention may have severe counterproductive (and normally unintended, even if not necessarily unexpected) effects. As Microsoft puts it:

---

[14] *Microsoft CFI*: Judgment of the Court of First Instance (Grand Chamber) of 17 September 2007 in Case T-201/04, Microsoft Corp. v Commission of the European Communities. (Available at http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:62004A0201:EN:HTML or http://www.microsoft.com/presspass/presskits/eucase/docs/T-201-04EN.pdf.) Even though the CFI endorsed the overall approach of the Commission, it should be mentioned that the Court did not approve the monitoring mechanism devised by the Commission (in fact, the CFI annulled Article 7 of Commission Decision 2007/53/EC, which established a monitoring mechanism, including a monitoring trustee).

[15] DEPOORTER & PARISI, *in* The Economics of Copyright: Developments.

[16] DARI-MATTIACCI & PARISI, *Substituting Complements*.

> We are subject to government litigation and regulatory activity that affects how we design and market our products. […] *The European Commission closely scrutinizes the design of high-volume Microsoft products* and the terms on which we make certain technologies used in these products, such as file formats, programming interfaces, and protocols, available to other companies. […] The Commission's impact on product design may *limit our ability to innovate* in Windows or other products in the future, *diminish the developer appeal of the Windows platform*, and *increase our product development costs*.[17]

However, given the possibility of achieving functional equivalence in software (see § 4.1) and given the economic incentive for platform incumbents to control complementary markets, which are of general interest for consumers, bundling arguably has the potential of excluding competitors and will likely reduce choice. Hence, a relevant question is: will less choice necessarily be a detriment to consumers? Given the benefits of standardization and the present state of knowledge in these fields of economics, no clear answer may be provided. That said, one must remember that the benefits of variety and competition in complementary markets are frequently difficult to estimate or even imagine[18] and they may likely surpass the benefit of standardization. Moreover, about bundling the Commission claims that the benefit that the bundling offers simply come from allowing "software developers and content providers […] to avoid the 'efforts of competition', which cannot constitute valid justification under Community competition law."[19] In other words, "[a]lthough, generally, standardisation may effectively present certain advantages, it cannot be allowed to be imposed unilaterally by an undertaking in a dominant position by means of tying."[20]

So, is there any paradigm to reconcile the aforementioned opposite views, which describe a quasi-monopoly platform either as an engine of innovation or as a brake to it?

2.1. Problems are the exception, not the rule

Antitrust, competition policy and industrial organization literature is rich in contributions that show if and when a dominant platform controller may act in an anticompetitive way.[21] A complete survey of this literature is far outside the scope of the paper at hand, but some background is nevertheless necessary. Hence, before going on proposing any kind of competition policy principles and remedies, I think that it is necessary to clarify that software markets – and other markets characterized by the likely presence of a "central" platform – need much fewer policy interventions than one may expect. In order to show that, the *ICE* paradigm proposed by Farrel and Weiser[22] is a very good starting point.

2.1.1. The ICE paradigm

The *Internalisation of Complementary Efficiencies* (*ICE*) paradigm is a post-Chicago generalization of the "one monopoly profit theory" of the early Chicago School thinking. The original "one monopoly profit theory"

---

[17] Microsoft Corporation, Annual Report to SEC (Form 10-K), Commission file number 0-14278, page 12, heading of *Item 1A. Risk Factors*. (Available at http://www.sec.gov/Archives/edgar/data/789019/000119312508162768/d10k.htm; last visited August 4, 2008.), pp 14-15. (Emphasis added.)

[18] See NET LE, *Microsoft Europe and Switching Costs*, 27 World Competition, 567—594 (2004), p. 592—593. "[O]nly when a market has been liberalised and consumers have benefited from it could one see how much they had suffered in the past. The spin-off of American Telephone and Telegraph (AT&T) in 1982 is a good example. Before the spin-off, AT&T argued that monopoly was important for a universal service sector, such as telecommunications, in order to exploit the 'economies of scale' and 'economies of scope' efficiently. After the spin-off, consumers benefited immediately and innovation flourished. As for AT&T, its tariffs decreased by 40 to 45 percent in the 1980s, but its volume increased annually by 6.7 percent. Had the spin-off been delayed, consumers would have suffered more. A competitive market would lessen innovation if the risks of sunk costs and free riding were too high. This is not thecase with bundling. If the incumbent were concerned about sunk costs, it would not give away the bundled product."

[19] *Microsoft CFI*, § 1128: "Because of the bundling of Windows Media Player with the 'ubiquity of the Windows monopoly', software developers and content providers who base their products on Windows Media Player do not need to convince users to install that player. By contrast, those who base their products on a third-party media player platform typically provide an incentive for users to install the necessary media player on their own computer, for example by including links for downloading the player through the Internet.". See also the *Commission's Microsoft Decision*, § 969.

[20] *Microsoft CFI*, § 1152.

[21] I will discuss, in particular, the approach of JOSEPH FARRELL & PHILIP J. WEISER, *Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age*, 17 Harvard Journal of Law & Technology, 85 (2003), but see also *supra* note 9.

[22] Id..

has been developed and applied in the works of Richard Posner, Robert Bork and others.[23] Its main point was that a monopoly platform would not have any interest in monopolizing an efficient and competitive complementary market and would instead have an incentive to stimulate an optimal level of competition and innovation in complementary markets (likely taking into account dynamic concerns as well)[24].

In its simplest possible version, "ICE claims that even a monopolist has incentives to provide access to its platform when it is efficient to do so, and to deny such access only when access is inefficient."[25] However, there are several very important exceptions to this "idyllic" situation and legislators and policy makers should be aware of them (without forgetting – at the same time – that the general rule is that platform controllers like to be at the centre of an innovative and dynamic environment). Several exceptions to the ICE paradigm are individuated by Farrell and Weiser and they may effectively guide the attention of antitrust authorities. In particular, eight exceptions are discussed in detail:[26] (1) Baxter's Law (i.e. in cases in which the platform is subject to regulation and the applications market is not, leveraging market power to the unregulated market can increase profits); (2) price discrimination (to perform a profitable price discrimination, the platform controller may need to control the application market as well, for instance because it is an efficient metering device); (3) potential competition (a complementary product may be a potential future competitor); (4) bargaining problems (access may be withheld because of transaction costs or for some strategic reasons); (5) incompetent incumbents; (6) option value (the platform controller may fear that granting access to some complementary markets could create a dangerous "precedent" and increase the likelihood of future regulation or antitrust intervention); (7) regulatory strategy (a variation on the previous exception, considering also other markets); and (8) incomplete complementarity (if applications are valuable without the platform, leveraging market power could be profitable despite ICE).

In principle, the traditional Chicago School findings regarding tying's uselessness as a leveraging tool are applicable only in cases in which complementary markets are perfectly competitive. This is surely not the case in the majority of software markets, were there are strong network effects and competition is far from perfect. However, as long as the market power of the platform controller is significant and the market power of producers of complementary goods is quite limited (allowing them to recoup their investment, but not to realize significant profits), it is reasonable to expect that platform owners will have more to gain than to loose from an environment of complementary innovations that work well. In any case, platform incumbents cannot "internalize everything", because their organizational complexity would become burdensome and they would risk innovating less and less and being again subject to Schumpeterian competition.

2.1.2. The platform controller as a regulation authority

The controllers of software platforms are likely to be the private undertakings with the higher interest in maximizing the overall value of "their" system, because they are the agents with the highest market power and can extract a big share of any surplus that may be generated in the system. In particular, being present in a complementary market is an important part of any platform owner's strategy. Indeed, "[a] foothold enables [the platform owner] to ensure the market for [the] complementary product stays competitive."[27] In other words, if a platform wants to play the role of a competition authority in complementary markets,[28] in order to ensure a decent qualitative level, competitive pressure to innovate, a good level of synergies and the achievement of all significant complementarities appreciated by customers, the platform owner may decide to compete in these markets using its own (complementary) product as a benchmark that other players will have to consider. At this stage, the compatibility between the platform owner's main product (e.g. Windows in the case of Microsoft) and competitors' products is full and the platform controller actually evangelizes its APIs and CPs. In fact, the decision to enter complementary markets may have been determined precisely because of the need to fully understand the needs of these markets and/or to show all the potentially complementarities between the platform's market and these other markets. The problem is that – once the

[23] See, for instance, the seminal work ROBERT BORK, *The Antitrust Paradox*, New York: Free Press (1978).

[24] Similar insights could be found also in the two-sided markets literature. See, in particular, J. C. ROCHET & J. TIROLE, *Two-Sided Markets: An Overview*, IDEI Toulouse working paper (March, 2004) (updated and published as J. C. ROCHET & J. TIROLE, *Two-Sided Markets: A Progress Report*, 37 RAND Journal of Economics, 645—667 (2006)).

[25] FARRELL & WEISER, *Modularity, Vertical Integration, and Open Access*, 89.

[26] Id., 105.

[27] FRANÇOIS LÉVÊQUE, *Innovation, Leveraging and Essential Facilities: Interoperability Licensing in the EU Microsoft Case*, 28 World Competition, 71—91 (2005), p. 83.

[28] *See supra* note 24.

platform owner has a decent product in the complementary market and given the fact that this market may be profitable in itself (as is frequently the case if the market is not perfectly competitive, something which may likely be excluded *a priori* when economies of scale are huge and network effects crucial) – a firm like Microsoft may be tempted, if it has some tools to do so, to leverage its market power in the complementary market. Moreover, this temptation will be higher if the control of this complementary market offers additional benefits (see above the exceptions to the ICE paradigm). For instance, in the case of workgroup servers, these operating systems are nowadays so technically similar to ordinary PC operating systems (running on more and more similar hardware) that the two markets could actually become a unique one in the future. Moreover, workgroup servers are a very promising market for one of the most dangerous of Microsoft's competitors, Linux.[29] The more Linux servers become widespread as web servers and workgroup servers, the more network administrators, developers and computer professional and experts in general become accustomed to them. Ultimately, this may decrease switching costs for ordinary users (and firms) considering to leave Windows in favor of Linux.

At the end of the day, if Microsoft had a button to switch-off competing servers, one could expect the software house to use it. Indeed, according to the Commission's reconstruction of the story, it did, even though in a more refined and subtle way:

> once Microsoft accounts for about 30 per cent of market share, it diminishes the level of disclosure of information to achieve interoperability. That makes rivals' products less valuable for consumers and lowers their sales. Buying a workgroup server equipped with a non-Microsoft operating system means a lower performance of the network because desktops are equipped with Windows. Customers also fear their equipment to be stranded if Microsoft completely ceases to disclose information. Microsoft's operating system benefits from the shift in market share. Developers of complementary software for servers expect Microsoft operating system for workgroup server to win in competition. They port their applications for it. Customers buy Microsoft's operating system because many more applications are available. As a result, the market tips in favour of Microsoft's operating system for servers, rivals' products are marginalized and competition is eliminated. Microsoft gains a better protection of its monopoly on the client PC operating system market. New entrants in this market will have to propose compatible software with Microsoft's server operating system. Moreover, potential competition between operating systems for server and desktop is eliminated.[30]

This is the now well-known dynamic story, making justice of the possibility of using refusal-to-deal (or disclose) as an anticompetitive tool.[31] Summarizing: In the short run, Microsoft received only advantages from interoperability, since the value of Windows was increased by the quality of any workgroup server operating system interoperable with it. However, later on potential competition (in particular form Linux low-end servers, using hardware similar to the one of ordinary computer) started to exist and the possibility of controlling a market, which was lucrative *per se* (being crucial for businesses), became apparent. Hence, Microsoft started leveraging its own Windows monopoly in the complementary market, with the aim of extinguishing competition (and reaping profits in the medium run).

Notice that, in such a context, the one-monopoly-profit theory would suggest that Microsoft would not use exclusionary policies. Instead, it could extract all the surplus from business users using an *ad hoc* licensing scheme, leaving to these users just enough informatics-related budged to buy server operating systems at a competitive price, barely sufficient to stimulate innovation in this market, leaving producers without any profit, once sunk costs are covered. Unfortunately, doing so – even if feasible – would have attracted (even more) the attention of antitrust authorities (because that would have required drastic price discrimination strategies, charging business users much more than home users, essentially for the same product). This may be one of the reasons why extracting the same surplus controlling the complementary market (the one of workgroup servers[32]) could have seemed safer from regulatory intervention (of course, this is another of the ICE paradigm's exceptions).

---

[29] In fact, the main difference between competitors such as Sun and Novel and Linux (in which also the "old" competitors of the server market increasingly invest) is that Linux increasingly and explicitly targets desktop PC users as a potential (and increasingly relevant) market, while former server producers concentrated their efforts on hardware platforms which were completely different from the ones of ordinary PC and well above the willingness to pay of individual computer users (e.g. mainframes).

[30] LÉVÊQUE, *Innovation, Leveraging and Essential Facilities*, p. 83.

[31] *See* Id., section II.

[32] Notice that, in order to fine tune the extraction of surplus from different categories of users, also controlling the complementary market of office suites would be very useful. Indeed, Microsoft does control and strongly protects this market as well.

Also, notice that, if Microsoft had simply modified Windows in order to artificially create complete incompatibility with competitors' server products, the reaction of users would have been very negative and all competition authorities around the world would have immediately jumped on this case and quickly decided to discipline this clearly exclusionary conduct. Instead, Microsoft kept Windows able to interoperate with any server using traditional communication protocols, but it also tied to Windows new pieces of software allowing for advanced features. This point is crucial and it qualified Microsoft's strategy. Microsoft was probably aware of the attention of antitrust authorities against the disruption of "existing levels of supply" (even if, here, we are talking about a special kind of "supply of interoperability information"). Hence, instead of degrading the existing level of interoperability, Microsoft added new features to Windows and it was regarding these new features, tied to Windows, that the software house used information-withholding in order to exclude competitors. This is why Microsoft's strategies may also be defined as bordering "predatory innovation:"

> [I]nnovation can be deemed exclusionary, and, therefore, violate antitrust law, whenever it prevents competitors' further innovation. That is to say that, when innovation becomes a means to block further innovation, and it entrenches the innovator in its own dominant position, the innovation may be considered exclusionary.[33]

Microsoft's strategy may very likely lie within this definition. In fact, producers of workgroup servers had always realized small client-side programs to be installed on Windows, Mac and other client operating systems in order to access to workgroup facilities (or, at least, to some advanced feature). A well-known example is Novell's NetWare solution.[34] Microsoft, instead, directly controlled Windows and had the possibility of making the client side of its technology strictly integrated into its client operating system. This lead to two outcomes: higher innovation, because of better synergies between clients and servers; and the expectation that these network-related functions of Windows actually worked on existing networks, so that users would complain with network administrators if they did not. Indeed, Microsoft's strategy did not reduce, in any way, its competitor's access to Windows. Instead, Microsoft "just" kept for itself new features, which it developed and which – apparently – users appreciated, because Microsoft's market share in workgroup operating systems grew and quickly surpassed 65 percent.[35] However, it is clear that – if Microsoft's competitors had had access to the new features bundled in Windows – innovation in the workgroup server market could have been even higher.

Overall, balancing Microsoft's interests, its competitors' interests and understanding which setting would create the higher level of innovation in the long run is far from clear: analyzing this issue requires some background about competition policy and antitrust issues, a background that the following sections try to synthetically provide.

3. Abusive conducts and the need for a manageable test

It has been observed that, in the field of competition policy exclusionary conducts are much more problematic than collusion to analyze.[36] One of the reasons of this complexity is precisely that, as I hinted, naked exclusion is quite rare, while naked collusion have been analyzed since the dawn of antitrust. Indeed,

> [i]n the vast majority of cases, exclusion is a result of conduct that has both efficiency properties and the tendency to exclude rivals. This is true of predatory pricing, exclusive dealing, tying, many types of bundling, and countless other forms of exclusionary conduct.
> The challenge in exclusion cases is how the law should treat conduct that has both efficiency benefits and exclusionary harm.[37]

This is why one may be tempted to opt for a complex balancing of the interests at play in order to solve the majority of competition policy cases involving exclusionary conducts. Indeed, several scholarly works[38] and

---

[33] *See* MARIA LILLÀ MONTAGNANI, *Predatory and Exclusionary Innovation: Which Legal Standard for Software Integration in the Context of the Competition v Intellectual Property Rights Clash?*, SSRN-id804985 (working draft) (September, 2005), p. 31.

[34] For details about NetWare, see http://en.wikipedia.org/wiki/Novell_NetWare (last visited August 3, 2008).

[35] *Commission's Microsoft Decision*, §493. For a much more detailed discussion (methodology for the measurement of market shares included), see section 5.2.2 of the *Commission's Microsoft Decision*.

[36] *See* A. DOUGLAS MELAMED, *Exclusionary Conduct under the Antitrust Laws: Balancing, Sacrifice, and Refusals to Deal*, 20 Berkeley Technology Law Journal, 1247 (2005), pp. 1248—1249.

[37] *See* Id., pp. 1248—1249.

[38] See, for instance, AHLBORN, et al., *Tying: A Farewell to Per Se Illegality*.

court decisions – including the recent European Microsoft case – seem to suggest some complex balancing tests in order to analyze the central issues of the paper at hand: tying and refusal to deal.[39] However, before completely embracing such an approach, one should consider that balancing tests are frequently very complex to apply and their outcome is not always easily predictable (indeed, the balancing is performed because of the complexity of the issue at hand). Moreover, when balancing is performed, weights – in the absence of universally accepted economic models to interpret reality – will depend on the economic policy agenda of the authority performing the balancing exercise. That is not only problematic in terms of legal certainty; it may also have the likely effect of creating discrepancies among the competition policy of different countries (something especially undesirable when real world markets are fully global, as in the field of software). In general, if the balancing exercise has to be repeated when design and strategic choices have to be taken, such an approach may also dangerously border the field of regulation, while "[a]ntitrust is better and more accurately understood to be a form of law enforcement, not regulation."[40] As observed by Melamed:

> Accepting the premise of antitrust as law enforcement not only is compelled by its nature, but also is important in order for antitrust to serve its contemporary substantive purposes. Antitrust rests on the premise that a decentralized market is most likely to create incentives for, and to take advantage of multiple sources of, creativity and entrepreneurship, thereby maximizing economic welfare. Antitrust thus presumes that government intervention should, as a general matter, be modest and should be undertaken only when the rules are clear and understandable so that uncertainty about the rules does not inhibit competitive and entrepreneurial forces that antitrust is intended to encourage.[41]

Understanding antitrust in this way implies not only that prohibited conducts should be clearly identified (turning on factual and, ideally, measurable issues), but also that the selection of antitrust rules should carefully take into account the principle of "administrability". This principle has two basic components: "the ability of courts and antitrust enforcement agencies to administer the rules after-the-fact"; and "the ability of businesses to know what conduct is permitted and what is prohibited."[42] In particular, about twenty years ago Areeda observed that "[n]o Court should impose a duty to deal that it cannot […] adequately and reasonably supervise".[43] Coherently with this approach, this paper will suggest that – when more than one remedy is available and all the remedies may presumably eliminate a threat to competition on the merits – the more administrable remedy should be chosen. Specifically, I will suggest that an unbundling order is more administrable than a obligation to disclose information, essentially because of the fact that the latter requires a quasi-regulatory activity in order to fix reasonable prices for disclosure.[44]

Given the impossibility of administering any balancing test from an omniscient point of view, this kind of test may have severe shortcomings (despite its abstract superiority).[45] Moreover, defendants themselves could not be expected to be able to understand whether some strategies, which are efficient from their point of view, create a more than counterbalancing amount of inefficiencies disfavoring competitors and ultimately users.[46] Hence, this paper tries to follow Melamed,[47] in adopting a middle path between the risks of "excessive confidence" implied by balancing tests – as the incentives-balancing-test proposed by the European

---

[39] See also W. KERBER, & C. SCHMIDT, *Microsoft, Refusal to License Intellectual Property Rights, and the Incentives Balance Test of the EU Commission*, (**2008**), presented at the EALE Annual Conference 2008. The authors interpret the "Incentives Balance Test" (that the European Courts seems to have applied in the Microsoft case) "as a test whether the specific IPRs of the dominant firm can be defended from the perspective of the economics of IPRs". Of course, if such an analysis could be applied case by case and on the basis of all the necessary information, this would lead to a first best; however one can hardly expect courts to be able to manage the complexity of real world cases (being at the same time consistent and predictable).

[40] MELAMED, *Exclusionary Conduct*, p. 1251.

[41] Id., p. 1251.

[42] Id., p. 1252.

[43] PHILIP AREEDA, *Essential Facilities: An Epithet in Need of Limiting Principles*, 58 Antitrust L.J., 841 (1989), p. 853: "No court should impose a duty to deal that it cannot explain or adequately and reasonably supervise. The problem should be deemed irremediable by antitrust law when compulsory access requires the court to assume the day-to-day controls characteristic of a regulatory agency."

[44] In principle, unbundling would also require the fixing of some price constraints (concerning the bundle and the separate products). However, as I will show (and specifically discuss in § 6), when a software platform and complementary products are under examination it may be sufficient to impose a "non-negativity" constraint on the price of the bundled complementary good (i.e., the price of the bundle must be higher or equal to the price of the unbundled platform alone).

[45] MELAMED, *Exclusionary Conduct*, pp. 1252—1255.

[46] Id., p. 1254: "From the perspective of the defendants, […] a balancing test would likely be either ignored, impose excessive transaction costs (a kind of tax on entrepreneurship), or result in excessive caution. There is little reason to expect that a balancing test would create optimal ex ante incentives for marketplace behavior."

[47] Id., p. 1255.

Commission in Microsoft IV[48] – and the minimalistic approach that would follow from the "extreme skepticism" of some of its critics (an example of such an approach would be the idea that intellectual property, secret included, provides an absolute shield against antitrust liability: see § 3.2). All that with the purpose of creating optimal (*rectius*: second best) *ex ante* incentives for marketplace behaviour. As for balancing tests, the ultimate goal of such an approach remains to "deter welfare-reducing conduct without reducing welfare-enhancing conduct". However – borrowing Melamed's example – this will be done proposing a given and certain "speed limit" in certain well defined "innovation roads". As for traditional road speed limits, it would be optimal to tailor a specific limit for each car and driver and to grant a plethora of exceptions depending on the circumstances. However, in order to create a reasonable amount of certainty and to make rules administrable, a unique limit has to be fixed and excuses should be quite narrow, with the burden of proof shifted on violators (if and when they decide not to respect certain limits).[49]

Having sketched this general and abstract background, the following paragraphs will discuss some more specific competition policy related concepts, which will be used in what follows. Then, these concepts will be applied in § 4, to analyze Microsoft antitrust cases, and in § 5, to derive an antitrust policy addressing the critical issues of the cases. All that will be done taking into account the principles of administrability and legal certainty.

3.1. Dominance and super-dominance

To discuss exclusionary practices (or any unilateral abuse of market power), the concept of dominance is crucial. The traditional definition of dominant position (or dominance) has been spelled by the European Court of Justice in the Hoffmann-La Roche case:

> [D]ominant position […] relates to a position of economic strength enjoyed by an undertaking which enables it to prevent effective competition being maintained on the relevant market by affording it the power to behave to an appreciable extent independently of its competitors, its customers and ultimately of the consumers.
> Such a position does not preclude some competition, which it does where there is a monopoly or a quasi-monopoly, but enables the undertaking which profits by it, if not to determine, at least to have an appreciable influence on the conditions under which that competition will develop, and in any case to act largely in disregard of it so long as such conduct does not operate to its detriment.[50]

The economic translation of this legal standard is far from obvious: for instance, no firm will really "behave to an appreciable extent independently […] of the consumers".[51] Even the most perfect monopolist will face a demand curve and act depending on it. However, despite the apparently impressive degree of market power that would seem to be needed to really integrate the Court's definition, the practical application of this standard translates in an analysis of market power and dominance is a concept, which is rich of degrees.[52] Indeed – despite the fact that a given market share is neither a necessary nor a sufficient condition to

---

[48] With *Microsoft IV* I refer to the *Case COMP/C-3/37.792 Microsoft*, that led to the *Commission Decision of 24.03.2004 relating to a proceeding under Article 82 of the EC Treaty*. The European CFI delivered its judgement on Microsoft's Appeal on the 17th of September 2007, substantially confirming the approach of the Commission (apart from some minor issues related to the Trustee established to monitor Microsoft's compliance).

[49] Unfortunately, the solution proposed by Melamed – the sacrifice test – is not especially well suited to deal with refusal to license cases. See MELAMED, *Exclusionary Conduct*. In fact, the author himself concludes that: "The sacrifice test is unambiguously superior to market-wide balancing in all cases in which the conduct does not raise rivals' costs, but rather excludes rivals only by reducing the defendant's prices or costs and/or increasing the value of his goods or services." (p. 1266) However, he admits that: "In refusal to deal cases, a balancing test would have the additional complication of requiring calculation of the costs to innovation incentives and dynamic efficiency of a duty to deal under the circumstances. A test that required such a calculation would plainly not be administrable by courts or firms. *The sacrifice test avoids this complication by incorporating the ordinary antitrust presumption that the dynamic benefits of encouraging innovation outweigh the costs of permitting firms to charge monopoly prices for their lawfully obtained monopolies.*" (p. 1267) [Emphasis added.] Making such a simplification here is unacceptable, because it creates an excessively high likelihood of false negatives, while there are way to adopt some clear rules which likely obtain a better result.

[50] Judgment of the Court of 13 February 1979 – *Hoffmann-La Roche & Co. AG v. Commission of the European Communities* –Case 85/76 at 38—39.

[51] See, for instance, MASSIMO MOTTA, *Competition Policy : Theory and Practice*, (Massimo Motta ed., Cambridge University Press. 2004), pp. 34—35.

[52] See IVO VAN BAEL & JEAN-FRANÇOIS BELLIS, *Competition Law of the European Community*, (Kluwer Law International. 2005), p. 119 (§ 2.24).

integrate a dominant position – firms holding a mere 40% market share may very well be considered dominant, where other elements are concord in signaling significant market power.[53]

In European competition policy, the concept of dominance is frequently accompanied by reference to the "special responsibilities" of dominant firms. As the Court of First Instance put it in Microsoft IV,

> whilst the finding of a dominant position does not in itself imply any criticism of the undertaking concerned, that undertaking has a special responsibility, irrespective of the causes of that position, not to allow its conduct to impair genuine undistorted competition on the common market.[54]

Even though US antitrust law seems to prefer references to "competition on the merits" as opposed to conducts impairing rivals,[55] concepts similar to the "special responsibility" of dominant undertaking are present also on the other side of the Atlantic, where (as in Europe)

> a monopolist is not free to take certain actions that a company in a competitive market (or even oligopolistic) market may take, because there is no market constraint on a monopolist's behaviour.[56]

Moreover, it must be noted that "[i]t is not only the fact of an undertaking's dominance that is relevant in misuse of market power cases under Art. 82 EC but also its extent".[57] In particular, I consider that the consequences of a disclosure obligation are not obvious and an excessively wide range of (presumably legitimate) market strategies may be hindered by such a legal requirement. For this reason, I will suggest to condition such an obligation (if it has to be established at all) to a finding of clear, uncontestable, stable and extensive dominance.[58] Hence, in this paper I will frequently talk about super-dominance, instead of simply referring to dominance. The concept of super-dominance made its first timid appearance already in the context of the famous Bronner case[59] (concerning the European application of the so-called "essential facility doctrine"), in the Opinion of the Advocate General Jacobs[60]:

> It seems to me that intervention of that kind [i.e. "requiring a dominant undertaking to supply the product or service or allow access to the facility"], whether understood as an application of the essential facilities doctrine or, more traditionally, as a response to a refusal to supply goods or services, can be justified in terms of competition policy only in cases in which the dominant undertaking has a genuine stranglehold on the related market. That might be the case for example where duplication of the facility is impossible or extremely difficult owing to physical, geographical or legal constraints or is highly undesirable for reasons of public policy. It is not sufficient that the undertaking's control over a facility should give it a competitive advantage.[61]

In that case, the reference to super-dominance was used in order to limit the applicability of any "essential facility" or similar doctrine to firms holding a true monopoly and not just any kind of dominant position. A more explicit reference to the concept of super-dominance came from Advocate General Fennelly's opinion in the *Compagnie Maritime Belge* (or *Cewal*) case. In this case, the dominant undertaking "had more than 90% of the market"[62] and engaged in likely exclusionary practices, hence the reference to super-dominance reinforced the need for intervention. According to Advocate General Fennelly, super-dominance – "a position of such overwhelming dominance verging on monopoly" – would create a kind of "special responsibility" exceeding

---

[53] See, for instance, MOTTA, *Competition Policy*, , pp. 35.

[54] *Microsoft CFI*, § 229. See also Case 322/81 Michelin v Commission [1983] ECR 3461, paragraph 57, and Case T-228/97 Irish Sugar v Commission [1999] ECR II-2969, paragraph 112.

[55] LePage's Inc. v. 3M, US Court of Appeals, Third Circuit. Filed March 25, 2003, 324 F.3d 141 at § 16:

§ 2 of the Sherman Act is violated whether "A monopolist wilfully acquires or maintains monopoly power when it competes on some basis other than the merits." See Aspen Ski case (Aspen Skiing Co. v. Aspen Highlands Skiing Corp., 472 U.S. 585, 605, n.32 (1985)), quoting 3 P. Areeda & D. Turner, Antitrust Law 78 (1978): "Thus, 'exclusionary' comprehends at the most behavior that not only (1) tends to impair the opportunities of rivals, but also (2) either does not further competition on the merits or does so in an unnecessarily restrictive way."

[56] *LePage's Inc. v. 3M*, 324 F.3d 141, 151 (2003).

[57] IAN EAGLES & LOUISE LONGDIN, *Microsoft's Refusal to Disclose Software Interoperability Information and the Court of First Instance*, 30 European Intellectual Property Review, 205—208 (2008), p. 205.

[58] *See § 4.7. Mandating disclosure* (and its sub-sections).

[59] *Oscar Bronner v Mediaprint Zeitung- und Zeitschriftenverlag GmbH and others* Case C-7-97 (1998).

[60] *Oscar Bronner v Mediaprint Zeitung- und Zeitschriftenverlag GmbH and others* Case C-7-97, [1998] ECR I-7817, [1999] 4 CMLR 112 [*Oscar Bronner*].

[61] Id., § 65.

[62] Opinion of Advocate General Fennelly, 29 October 1998 – *Compagnie Maritime Belge Transports SA, Compagnie Maritime Belge SA and Dafra-Lines A/S v Commission of the European Communities*, joined cases C-395/96 P and C-396/96 P, § 144. (Available at http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:61996C0395:EN:HTML; last visited August 3, 2008.).

even the *ordinary* level of "special responsibility" of a "normal" dominant undertaking (having significant market power, but not being a monopolist).[63] For instance, in the case discussed by the Advocate General, instead of being liable of an antitrust violation just in case prices were below total average cost (the standard test for predation), a super-dominant undertaking would be liable also in case of targeted, selective price policies, making sense just because they were "designed to eliminate" a dangerous competitor.[64]

To summarize, one could say that the degree of special responsibility increases with the degree of dominance. A similar approach seems to be confirmed by the European Commission and by the European Courts, even if they used the wording "quasi-monopoly" instead of "super-dominance".[65]

I do not want to discuss here whether the concepts of dominance and/or super-dominance are well defined: I will discuss cases in which incumbents are surely super-dominant, and that will simplify this part of the analysis. Indeed, in software markets it happens quite frequently that a market leader has a market share well above 80 percent, because of several well-known economic phenomena: in particular, network effects (direct and indirect), switching costs and economies of scale. Thus, the whole point of this paragraph is to make clear that the discussion of this paper applies to undertaking stably holding a super-dominant position (and, as a rule of thumb, market shares above 80 or even 90 percent). With "*stably* holding a super-dominant position", I mean that the incumbent has been dominant for more than one main "version" of its product. This condition is not respected by any killer application and not even by a successful new platform. The kind of approach that I follow in this paper assumes the existence of a super-dominant platform working as the *de facto* standard for a certain market, that has done so for a while and – what is more important in high-tech markets – that is generally expected to dominate for quite some time into the foreseeable future.

Since the Microsoft case provides the main empirical ground of this paper, I point to the fact that, for instance, Microsoft has been super-dominant in the client PC operating system market (at least) with Windows 95, Windows 98, Windows XP and Windows Vista.[66] As found by the European Commission,

> Microsoft's dominant position on the client PC operating systems market exhibits […] 'extraordinary features', since, notably, its market shares on that market are more than 90% […] and since Windows represents the 'quasi-standard' for those operating systems.[67]

Moreover, this market has been found to be a "relevant market" for antitrust purposes in several antitrust decisions on both sides of the Atlantic.[68]

3.2. Intellectual property is not an absolute excuse

Another problematic question concerns the relationship between intellectual property and competition policy. Some scholars argued in favor of the so-called "*inherency doctrine*, according to which intellectual property law defines the scope of protection [offered by the legal system to innovators] and competition law must not interfere".[69] However, I will just briefly discuss such an approach, since I consider that rulings on both sides of the Atlantic already clarified, as a matter of law, that the exercise of intellectual property does not offer an absolute excuse to any antitrust violation. Streams of ink have been spilled about these issues, but some precedents still hold. In Europe, the *Volvo* decision[70] is a fundamental precedent, starting a stream

---

[63] Opinion of Advocate General Fennelly (*supra* note 62), § 137.

[64] Opinion of Advocate General Fennelly (*supra* note 62), § 137-138.

[65] See, for instance, Microsoft CFI decision, § 775: "the Commission was correct to find at recital 787 to the contested decision [Microsoft IV] that when Microsoft had responded to the letter of 15 September 1998 it had not sufficiently taken into account its *special responsibility* not to hinder effective and undistorted competition in the common market. The Commission was also correct to state, at the same recital, that that *particular responsibility derived from Microsoft's 'quasi-monopoly'* on the client PC operating systems market." (Emphasis added.)

[66] Not to mention various combinations of DOS + Windows 3.x and other relatively "minor" editions of Windows, such as Windows ME.

[67] *Microsoft CFI*, § 387.

[68] With *Microsoft III* or *Microsoft US case* I refer to the famous US antitrust case, which risked to lead to the dismembering of Microsoft, following J. Jackson's ruling 87 F.Supp.2d 30 (D.D.C., 2000). In appeal, the case was vacated and remanded, with ruling 253 F.3d 34 (C.A.D.C., 2001). The case has been ended by the *Consent Decree* ratified by J. Kollar-Kotelly, with ruling 231 F.Supp.2d 144 (D.D.C., 2002). Consider also that with *J. Jackson's findings of fact* I refer to ruling 84 F.Supp.2d (D.D.C., 1999). For a vivid description of the crucial economic issues of the case, see DAVID S. EVANS, et al., *Did Microsoft Harm Consumers? Two Opposing Views*, (David S. Evans ed., AEI Press. 2000).

About *Microsoft IV* see *supra* note 48.

[69] See DREXL, *IMS Health and Trinko* for a broader discussion.

[70] Case 238/87.

of decisions leading to the conclusion that "despite the legality of the refusal to license as such, the exercise of an exclusive right might constitute an abuse".[71] Similarly, but even more clearly, in the US the D. C. Circuit Court of Appeals quite recently (in Microsoft III)[72] defined as "border[ing] upon the Frivolous" the theory according to which "if intellectual property rights have been lawfully acquired, […] their subsequent exercise cannot give rise to antitrust liability." In fact, according to the Court, "[t]hat is no more correct than the proposition that use of one's personal property, such as a baseball bat, cannot give rise to tort liability."[73] These decisions seem to disprove the *inherency doctrine*, even if they confirm that – the ordinary use of intellectual property being the one of excluding some users and competitors from accessing a certain immaterial good – a refusal to license is normally legitimate (as it is normally legitimate for an undertaking to keep his own material property for his exclusive use).[74]

3.2.1. Overcoming the new product test

Hence, intellectual property cannot provide a complete shield against antitrust liability, nor can simple secrecy, which is surely deserving even less protection, as it concerns similar "immaterial objects", but it is unilaterally created, instead of granted (for a limited period) by the legal system (possibly in exchange for disclosure). As the Court of First Instance observed,

> there is no reason why secret technology should enjoy a higher level of protection than, for example, technology which has necessarily been disclosed to the public by its inventor in a patent-application procedure.[75]

That having been said, the general rule governing free markets is that an undertaking may choose its own commercial partners and that it is free to design its products as it likes. Certainly, this rule knows several qualifications in some "exceptional circumstances" (using a wording adopted by the European Court of Justice). At the borders between competition and intellectual property, the general principle governing these exceptions should be, according to Drexl, that "the 'exceptional circumstances' […] have to be found in the impossibility of other companies to compete by substitution."[76] In other words (following an approach similar to the one proposed by Weiser[77] and discussed in the first paper of this dissertation) the law should encourage competition between standards and/or for a standard as long as this is possible, but it should enforce competition within a *de facto* standard, once it has been clearly and stably established in the market.[78]

It must be observed that the "exceptional circumstances" under which there may be a duty to license/disclose have been initially defined in a quite narrow way by European Courts. In *Magill* and *IMS*,[79] the relevant question was whether a dominant incumbent limited "production or markets to the prejudice of consumers", in particular preventing the appearance of a "new product", for which a (potentially) significant demand existed. As a whole, the test laid down in Magill and IMS may be summarized in the following way:[80]

> four cumulative conditions were laid down, which, if met, would see intellectual property owners held liable [of an abuse of dominant position]. This would occur if the refusal to licence:
> (i) related to a product or service indispensable to the exercise of an activity on a neighbouring market;
> (ii) was of such a kind as to exclude any effective competition in that market;

---

[71] DREXL, *IMS Health and Trinko*.

[72] *See supra* note 68.

[73] Id., p. 33.

[74] *See*, for instance, *Microsoft CFI*, § 691: "It must be borne in mind that […] the Community judicature considers that the fact that the holder of an intellectual property right can exploit that right solely for his own benefit constitutes the very substance of his exclusive right. Accordingly, a simple refusal, even on the part of an undertaking in a dominant position, to grant a licence to a third party cannot in itself constitute an abuse of a dominant position within the meaning of Article 82 EC. It is only when it is accompanied by exceptional circumstances such as those hitherto envisaged in the case-law that such a refusal can be characterised as abusive and that, accordingly, it is permissible, in the public interest in maintaining effective competition on the market, to encroach upon the exclusive right of the holder of the intellectual property right by requiring him to grant licences to third parties seeking to enter or remain on that market."

[75] *Microsoft CFI*, § 693.

[76] DREXL, *IMS Health and Trinko*, p. 805 ff..

[77] FARRELL & WEISER, *Modularity, Vertical Integration, and Open Access*.

[78] See below, in particular § *4.7. Mandating disclosure*.

[79] Radio Telefis Eireann (RTE) and Independent Television Publications Ltd (ITP) v. Commission of the European Communities (*Magill*), joined cases C-241/91P and 242/91, 1995 ECR I-743; IMS Health GmbH & Co. OHG v. NDC Health GmbH & Co. KG (*IMS*), case C-418/01, ECR I-5039 (2004).

[80] *See* EAGLES & LONGDIN, *Microsoft's Refusal to Disclose*, p. 207.

(iii) prevented the appearance of a new product for which there was potential consumer demand; and
(iv) could not be objectively justified (the onus here being on the refuser).

Essentially, such a test was equivalent to the "standard" test forming the European equivalent of the essential-facility-doctrine,[81] as developed in the *Bronner* case[82], with the addition of condition (iii). Hence, for a while, preventing the appearance of a new product has been considered as "The Exceptional Circumstance" allowing the essential facility doctrine to be applied in cases were the licensing of intellectual property (and possibly trade secrets[83]) was at stake. However, the Court of First Instance observed in its Microsoft Decision that

> The circumstance relating to the appearance of a new product, as envisaged in Magill and IMS Health […], cannot be the only parameter which determines whether a refusal to license an intellectual property right[84] is capable of causing prejudice to consumers within the meaning of Article 82(b) EC. As that provision states, such prejudice may arise where there is a limitation not only of production or markets, but also of technical development.[85]

Indeed, as the Court recalls in its ruling, the "new product test" is not explicitly mentioned under Article 82(b) EC, while the general principle according to which dominant undertakings should not limit production, markets and technical development is.[86] Hence, the "new product case" would just be one of the possible examples of such a general principle. The general principle essentially requiring that, because of the exceptional circumstances of a case, access to a certain set of information (and the right to use it, in case this information is protected by intellectual property) is necessary in order to compete in a given market.[87]

The problem is that, while the narrow "new product test" had quite predictable consequences as long as it was applied to cases like the Magill one, the "incentive balancing test" proposed by the Commission is much more problematic as far as administrability is concerned. Frankly, the "incentive balancing test" borders the imposition of a Kantian categorical imperative on dominant undertakings:[88] when competition by substitution cannot guarantee third parties a level playing field, the Commission seems to require that dominant firms take their licensing choices in looking not only at their own self interest in an given setting, but considering also the impact of their actions on the industry as a whole. In other words, it seems to me that firms do not have to find "just" an "objective justification" for their own refusal, but a justification that could "become a universal objective justification":

> The major objective justification put forward by Microsoft relates to Microsoft's intellectual property over Windows. However, a detailed examination of the scope of the disclosure at stake leads to the conclusion that, on balance, the possible negative impact of an order to supply on Microsoft's incentives to innovate is outweighed by its positive impact on the level of innovation of the whole industry (including Microsoft). As such, the need to protect Microsoft's incentives to innovate cannot constitute an objective justification that would offset the exceptional circumstances identified.[89]

Indeed, Microsoft's description of this test is not so far from the reality:

> [T]he Commission considered that a refusal to communicate information protected by intellectual property rights constituted an infringement of Article 82 EC if, all things considered, the positive impact on the level of innovation in the whole industry outweighed the negative impact of the dominant undertaking's incentives to innovate.[90]

---

[81] Notice that, however, the European Commission and the European Courts do not typically refer to the "essential facility doctrine" using this wording, preferring to talk about cases, which refusal-to-deal constitutes an abuse of dominant position.

[82] Oscar Bronner GmbH & Co. KG v. Mediaprint Zeitungs-und Zeitschriftenverlag GmbH & Co., Case C-7/97, [1998] E.C.R. I-7791, [1999] 4 C.M.L.R. 112 (*Bronner*).

[83] See footnote 84.

[84] The licensing/disclosure of trade secrets is clearly included by the Court. *Rectius*, it is clear, in the ruling, that in cases in which there is a duty to license intellectual property rights there is also a duty to license trade secrets. It is, however, less clear whether there may be less stringent "exceptional circumstances" able to force the licensing of secrets, but not of intellectual property. What can be said is that there are no elements in the *Microsoft CFI* ruling to argue that this is the case.

[85] Microsoft CFI, § 647.

[86] Microsoft CFI, § 643.

[87] *See*, for instance, EAGLES & LONGDIN, *Microsoft's Refusal to Disclose*, p. 208.

[88] IMMANUEL KANT, translated by James W. Ellington (1993), *Grounding for the Metaphysics of Morals*, 3rd ed., Hackett (1785), 30: "[a]ct only according to that maxim whereby [they] can at the same time will that it should become a universal law."

[89] *Commission's Microsoft Decision*, § 783.

[90] *Microsoft CFI*, § 669.

In other words, applying this test, dominant firms should substitute to their own self interest the overall interest of their industry (in a rather broad sense, because they should consider at least their own relevant market and the complementary ones). To be fair, the Commission also stressed that Microsoft's incentives to innovate in the field of protocol (and API) specifications (and design in general) do not rely only (nor mainly) on its freedom to manage (and possibly license) these protocols (and APIs) as it likes.[91] And that is certainly true, since despite any obligation to disclose its specifications, Microsoft could likely have incentives to innovate in interoperability specifications, just to "sell" the rest of its products (possibly including interoperability implementations), taking advantage of interoperability.

Of course, as the Commission stressed, Microsoft's argument "to show that the obligation imposed on it to disclose the interoperability information would have a negative impact on its incentives to innovate […] were purely theoretical and wholly unsubstantiated".[92] However, so were the Commission's claims that the opposite is true. Indeed, it is the difficulty in determining how much Microsoft's incentives to innovate would be reduced by a disclosure obligation on specifications that shows how the Commission's balancing test is irremediably flawed. The problem is not that the Commission necessarily applied the test in a wrong way. Indeed, if there was an omniscient planner accepting bets on this issues, I would bet on the Commission's side. However, the problem is that such a test cannot guide the behavior of competition agencies in a sufficiently predictable way. Hence requiring dominant undertakings to unbiasedly apply the test to their own disclosure decisions is, at least, bizarre. Probably, the only way "to be safe" would be to disclose all interoperability specifications. That could be a socially desirable outcome, but I suggest that this could be reached by following a clearer rule, imposing a generalized disclosure obligation only where quasi-perfect monopoly (i.e. a super-dominant position) exists and, possibly, offering an alternative to such a disclosure obligation, when dominant undertakings decide to follow a clear-cut modularity principle.

To be fair with the Commission, another point is worth mentioning. In fact, the Court of First Instance highlighted that the "incentive balancing test" is not a "new test" concerning the possibility of refusing the licensing/disclosure of intellectual property protected works/secrets.[93] According to the Court, point (iii) of the Magill test has "simply" been discussed by the Commission, not using a single test, but convincingly showing that Microsoft's competitors would have produced new and innovative workgroup servers, if only Microsoft had given them the possibility of doing so, disclosing a certain amount of indispensable information.[94] In other words, point (iii) has been respected, showing with theoretical reasoning and empirical examples that Microsoft's refusal-to-deal "limited technical development to the prejudice of consumers".[95] It is only at that point that the Commission evaluated whether Microsoft's justifications were sufficient to overcome these "exceptional circumstances".[96] However, also the impression that the "incentive balancing test" is a test for the acceptability of a justification from a dominant undertaking is incorrect, according to the Court. To reach this result, the Commission mentioned several facts, including the empirical evidence concerning industry practices, "establishing that the disclosure of interoperability was widespread in the industry concerned".[97] Overall, the Court concluded that what Microsoft and several commentators called the "incentive balancing test" is just a construct "based on a single sentence",[98] which is not particularly crucial for the Commission's argumentation.

Unfortunately, the Court's argumentation is not completely convincing. Indeed, the majority of scholars[99] having commented the Commission's Decision had – as Microsoft – the impression that the "incentive balancing test" was quite a central point in the Microsoft Decision, essentially summarizing points (iii) and (iv) of the previous test or, at least, being crucial to reject an undertaking's justifications under point (iv). Denying that, and explicitly excluding that the "new product test" should be interpreted in a narrow way, but without offering any clear alternative, the Court leaves us with even less legal certainty. At the end of the day, the Court seems to say that the Commission has been convincing about the fact that, in this case,

---

[91] *Microsoft CFI*, § 685.
[92] *Microsoft CFI*, § 680.
[93] *Microsoft CFI*, § 704.
[94] *Microsoft CFI*, §§ 704—710.
[95] *Microsoft CFI*, § 709.
[96] *Microsoft CFI*, § 710.
[97] *Microsoft CFI*, §§ 710 and 702 in particular.
[98] The sentence reads as follows: "[A] detailed examination of the scope of the disclosure at stake leads to the conclusion that, on balance, the possible negative impact of an order to supply on Microsoft's incentives to innovate is outweighed by its positive impact on the level of innovation of the whole industry (including Microsoft)" (*Microsoft CFI*, § 704).
[99] E.g. LÉVÊQUE, *Innovation, Leveraging and Essential Facilities*; EAGLES & LONGDIN, *Microsoft's Refusal to Disclose*.

"exceptional circumstances" applied and that disclosure/licensing was a better solution for the market than secrecy, to the benefit of the technological progress (including the one potentially generated by Microsoft). Frankly, saying that – despite what the Court may argue – means precisely that the incentive balancing test exists and that it may be an alternative to the "new product test" to pass step (iii) of the analysis of a refusal-to-deal case. Moreover, despite the fact that such an incentive balancing does not completely substitute point (iv) of the analysis, the result of the balancing may be considered to reject an undertaking's justifications, when these are not particularly compelling and seem to be just based on a presumed reductions of the undertaking's incentives to innovate. Finally, accepting the Commission's analysis, the Court implicitly admitted that the balancing test may be passed just gathering a sufficient amount of overall convincing theories and anecdotal empirical evidence.

Admittedly, in analyzing issues as complex as the ones presented in the Microsoft case some simplifications are required (as I will discuss further, summarizing the Microsoft case below). However, it has already been shown[100] that the Commission's analysis of the case makes excessive simplifications, which are stronger than the ones that will be needed in order to accept the approach suggested in this paper (or, at least, not weaker).[101] In particular, the Commission argues that an order to disclose interface information would increase rivals' incentives to innovate, probably without reducing Microsoft's incentives too much or, at least, reducing them in a less than balancing way.[102] Lévêque summarize the economic reasoning of the Commission as follows:

> In a first step, the Commission points out that the non-disclosure impedes Microsoft competitors from capturing the benefits of their innovation in workgroup server operating systems. In so far as compatibility with Windows is a killer characteristic of this product, it does not pay to improve other characteristics without access to information on interface. In other terms, if firm B cannot get access to innovation 1, firm B does not make innovation 2. By contrast, '[I]f Microsoft's competitors had access to the interoperability information that Microsoft refuses to supply, they could use the disclosures to make advanced features of their own products available'. In a second step, the Commission analyses the changes in incentives to innovate for Microsoft itself. It argues that Microsoft's incentives to innovate in the workgroup server operating system will be reduced in case of non-disclosure. It invokes the absence of competitive pressure as a cause of this decline. Then, the Commission considers Microsoft's incentives to innovate to improve interface. It stresses that 'there are good reasons to believe' that Microsoft has interest to do it, for Microsoft sells both PC and workgroup server operating systems. It concludes, 'It is dubious whether an order to supply in this case would have any negative impact on Microsoft's incentives to innovate'.[103]

As Lévêque showed, the second part of the Commission's reasoning is "questionable":

> Firstly, whether a monopoly or competitive firm is better placed to innovate remains a controversial question amongst economists. For the Commission a market structure dominated by a monopoly seems to always provide less innovation than a market structure with several firms in competition. Secondly, incentives to innovate on interface necessarily decrease where firm A is not alone in producing the complementary product. The reason is that it can no longer expect to capture all the value of its innovation. It is true that Microsoft, even with the compulsory licensing, will continue to have an interest in innovating on interface but it is also true that it will have less of an interest to do so. Therefore, Microsoft's incentives to innovate in interface decrease with compulsory licensing.[104]

According to the author, it is only under a very strong assumption and assuming that an incentive balancing test must be performed that the Commission may be said to be right:

> In so far as the essential facility owner has to prove the existence of an objective justification, the Commission is right: it is not evident that the compulsory licensing will decrease Microsoft's incentives to innovate more than it will increase the rival's incentives to innovate.[105]

Looking back at the four steps in the test to analyze refusals-to-deal, the fact of putting the burden of proof on Microsoft seems to be coherent with the fact that – despite what the Court said – step (iv) of the test has

---

[100] *See* LÉVÊQUE, *Innovation, Leveraging and Essential Facilities.*
[101] See in particular § *4.7. Mandating disclosure.*
[102] See, in particular, *Commission's Microsoft Decision*, § 783 (and, in general §§ 709 ff.).
[103] LÉVÊQUE, *Innovation, Leveraging and Essential Facilities*, pp. 78—79.
[104] Id., p. 79.
[105] Id., p. 79.

been substituted or, at least, integrated by the new "incentive balancing test".[106] Indeed, some kind of "reversal of the burden of proof" is, in my opinion, necessary to deal with cases as complex as that of Microsoft. My objection is just that a clearer rule should be set, saying when this reversal of the burden of proof takes place and the principle of disclosure (and of modularity, as I will discuss) must be respected. In other words, this rule may allow undertakings the principles of disclosure and modularity, but only as long as they are able to prove (with the burden of proof resting on them) the sub-optimality of such an approach in the case at hand. (Clearly, this reversal of the burden of the proof may be problematic in some circumstances, but the same is true for a driver speeding toward the hospital because of an emergency.) Indeed, some kind of reversal of the burden of the proof seems to be already at play in the Commission's Microsoft Decision, and such an approach has been approved by the Court of First Instance.

3.3. Who bears the cost of creating network effects?

Another point concerning the shaky borders between competition policy and intellectual property may be worth mentioning, because it should be considered to better analyze incentives to innovate. In fact, whether a cost have been incurred by the company, having realized a certain product, or by its customers and/or by producers of complementary products matters in determining if there is an obligation to deal and license a certain facility (and in determining whether it may be considered "essential").

Clearly, switching costs have to be taken into account in determining whether a certain asset is or not indispensable to compete in a market.[107] In a way, switching costs are customers' investment that contributes to the creation of an assets' "indispensability". Moreover, the direct participation of customers in creating the indispensable assets has to be taken into account. For instance, the IMS Health case[108] concerned the possibility of reproducing a database structure, which became a *de facto* standard for the customers in the relevant market under examination. In that case, not only would adopting a database with a different structure have required significant switching costs,[109] but the IMS database customers had participated significantly in the creation of the structure of the database and that did matter in deciding the case.[110]

Similarly, the value of a dominant platform is frequently related not only to the investments made by its own developer, but also to the investments made by users and developers of complementary products. This is primarily related to users' and complementary developer's sunk costs (in the form of learning costs, network effects which are related to the platform, and other switching costs). But also to the fact that the feedback of these agents progressively stratified and contributed to the design of the platform may matter. Following the reasoning applied in the IMS case, this fact may be taken into account while deciding whether a disclosure obligation may be imposed on software platform incumbents.

The previous observation is not crucial under the approach that I will propose, since I will try to avoid tests that are too complex to manage. However, especially if one decides to follow an "incentives balancing approach", the role of third parties in building the system centered around the platform should be considered carefully. Indeed, in justifying a refusal to deal as determined by the need to maintain incentives to innovate, a super-dominant undertaking should not be allowed to claim as its own some investments, which have instead been cumulated by third parties. At the same time, if third parties received an incentive to invest in platform-specific goods from the dominant undertaking – for instance in the form of cross subsidization among various sides of a platform (e.g. Microsoft organizes courses for developers, using resources "extracted" from

---

[106] In *Microsoft CFI*, issues related to the burden of the proof are mostly discussed at §§ 688 ff.

[107] *IMS* (*supra* note 89) opinion, § 84—85: "to persuade [potential customers] to acquire [a new product], the entrant would have to offer the consumers particularly favourable terms with the risk that the investment made would not be amortised. It must therefore be deduced that […] [switching costs] are elements to be taken into account in establishing whether or not there are obstacles of a technical [including network effects], legislative [including intellectual property rights] or financial nature which may make it impossible […] for any [entrant] to create [its product] possibly in conjunction with [the incumbent's product].".

[108] *IMS* (*supra* note 89).

[109] *IMS* Decision 2001/165/EC, § 119: despite the fact that certain pharmaceutical companies "were unable to estimate switching costs […], [t]he mentioned costs vary from 40,000 DM to 1.85 million DM, around 30 percent of the annual budget for regional sales data for a large pharmaceutical company. For small and medium size companies, they represent from 25 to more than 100 percent of the annual budget for regional sales data."

[110] *See* LE, *Microsoft Europe*, p. 576. and *IMS*, ECJ judgment of April 29, 2004, § 30: "The degree of participation by users in the development of [a platform or a cross-platform element], particularly in terms of cost, on the part of potential users in order to purchase [rival products] presented on the basis of an alternative [platform or cross-platform] are factors which must be taken into consideration in order to determine whether the [copyright] protected [platform or cross-platform] is indispensable to the marketing of [products] of that kind."

Windows users) – competition authorities should remember that this cross subsidization activity may be possible only as long as the platform owner may exercise a certain degree of market power[111]. Overall, I doubt that cross subsidization requires the existence of a quasi-perfect monopoly, but this and similar issues highlight how complex it may be to take into account all the relevant aspects of a case by case incentive balancing test.

4. Teachings from the Microsoft cases

During the last twenty years (or more), Microsoft's market share in the market for PC operating systems remained stable above 80 percent, and above 90 percent over the last ten years.[112] This prolonged dominance provides a huge natural experiment and significant evidence regarding the possible strategies, which a dominant undertaking, controlling a software market platform, may put into place in order to strengthen its market position, but also in order to hold it for a significant period. Indeed, during this period the software industry flourished and – at the same time – Microsoft was the object of several antitrust investigations. Hence, Microsoft cases on both sides of the Atlantic provide hints concerning the key element of exclusionary innovation practices. At the same time, the history of Microsoft also confirms the basic insight of the ICE paradigm (and of two-sided markets models),[113] according to which software market incumbents do not have any interest in stifling innovation, apart from some targeted interventions, able to eliminate potential competitive menaces.

A first element to take into account is that platform incumbents are frequently late (at least second) comers in complementary innovation markets (this language presupposes that the incumbent's controlled platform is a central and necessary element of a complex system, composed by several pieces of software, running on the same set of hardware – a single PC – or on more computers connected in a network). That happened at the time of the introduction of browsers and Internet distributed applications, when Netscape and Java preceded similar Microsoft's technologies (for instance, Internet Explorer and .Net). Something comparable happened again in the field of media players, where the success of RealNetworks' and Apple's products predated the one of Microsoft's Media Player.[114] All that is discussed further in § *4.1.1. Platform leaders as late comers in new complementary markets.*

Another thing one can learn from these cases is that tying a piece of software to a dominant platform could guarantee its widespread availability. In principle, this may or not be sufficient to guarantee that it will be used, but it is a crucial step, since the fact that a piece of software is already present on users' PC implies that complements and/content producers may "recall" it at no cost either for them or for their users. Hence, indirect network effects may be created by simply tying a given piece of software to another widespread piece of software. Moreover, it must be considered that not all users are the same: even though advanced users may

---

[111] Indeed, this is why some scholars found that a monopoly platform may be preferable to competition in two-sided markets, at least under some (restrictive) conditions. *See*, in particular, J. C. ROCHET & J. TIROLE, *Platform Competition in Two-Sided Markets*, 1 Journal of the European Economic Association, 990—1029 (2003).

[112] For evidence about the first decade of this period, see *J. Jackson's findings of fact* (*supra* note 68), § 35 ff., in the US Microsoft Case. For later periods, see the *Commission's Microsoft Decision*, § 432: "Microsoft's extremely high market shares have not come about recently. In 1996, Microsoft had a market share of 76.4%, and since 1997 has held market shares of consistently over 80%, and of over 90% since 2000."

[113] See footnote 28 and the accompanying text.

[114] Of course, incumbents arriving first to market and proposing products at the frontier of research exist. However, this is not a case that should worry us too much, nor is their existence worrying for competition policy. Indeed, as long as they control a central part of a platform, they may represent a *de facto* standard for a given industry, with some benefits from the industry as a whole, as happened with IBM in the mainframe industry over several years: "[N]o one could beat IBM to market with a new product line. If a competitor tried to invade its space ahead of IBM, it could never be sure that IBM's next operating system release would be incompatible with its product, especially if the product was one IM wanted for itself. Competitors had no choice but to reverse-engineer IBM products only after they became available, and therefore were comdemned always to be second to market. And by the time competitive plug-compatible products decame available, IBM was usually already moving on to the next product generation.". See Charles H. Ferguson & Charles R. Morris, *Computer Wars: How the West Can Win in a Post IBM World*, 15—16 (1993), quoted by JONATHAN BAND & MASANOBU KATOH, *Interfaces on Trial — Intellectual Property and Interoperability in the Global Software Indusry*, (Jonathan Band ed., Westview Press First ed, Boulder, Colorado. 1995), p. 24. Obviously, once a given technology becomes "commoditized", as happened with personal computers, the leadership of the incumbent could fade, but this just means that it should re-focus its activities on more cutting-edge fields (as IBM did, in selling its ThinkPad laptop brand to the Chinese producer Lenovo, but continuing to compete fiercely in the server market: See http://en.wikipedia.org/wiki/Lenovo for a detailed timeline). But this just means that supra-normal profits cannot be extracted for very long periods of time, unless the pace of innovation is significant: hardly a worrying scenario.

neglect the cost of looking for and installing new software (indeed, they may even enjoy the process), the majority of users may dislike installing new pieces of software and/or not having a PC ready to perform the majority of tasks. Hence, tying a piece of software to a dominant platform may actually increase the use of this piece of software creating also direct network effects. That means that – *ceteris paribus* (quality included) – a piece of software tied to a dominant product will most likely conquer the market: as I will discuss, this result is more problematic than it looks at a first glance (see § 4.1).

Finally, it must be noted that previous observations neglect two other players of this game. Original Equipment Manufactures (OEMs) and/or other professional intermediaries (e.g. ICT departments, network administrators and so on) may act as consultants for less advanced users when it comes to choosing which software are installed on a user's PC. Hence, when it is possible to uninstall a certain piece of software, its distribution with a dominant platform does not preclude the possibility that also "dummy users" actually receive a personal computer where this piece of software has been substituted with another one or, at least, where an alternative program has been set as "default application". Similarly, ISP and content providers are part of the network externalities creation network. Thinking about the media player industry, it should probably be added that browsers producers play a relevant role here as well (making it easier or harder to realize and install some add-ons and various plug-ins that may be needed to play certain contents). Hence, OEM, ISP and similar actors are all agents to be "taken on board" by new technology developers. Indeed, the fact that the platform controller is frequently a late comer means that it has the difficult task of convincing them, overcoming switching costs (learning cost, indirect network effects, etc.). But the platform controller may also use several shortcuts, from direct agreements with these agents (explicitly leveraging its market power in terms of discounts or exclusionary practices) or, more subtly, to prevent these potential users' "consultants" from uninstalling its own pieces of software.

4.1. Functional clones and late comers

In software markets it is almost always possible to create a functional clone of someone else's product. It is a well known pillar of intellectual property law – which I extensively discussed in the first paper of this dissertation – that copyright does not protect ideas, but only the original expression of a piece of software. Hence, if you have deep enough pockets, you can produce a more or less perfect functional clone of any computer program. However, the weakness of copyright protection is not creating severe market failures. This is due, amongst other things, to some of the principles I discussed in the second paper. To summarize, the functional cloning of someone else's successful software is likely to be economic suicide, unless you are also able to significantly improve the first comer's product. And this is so because the incumbent will still be in the market, with zero marginal costs (so able to quite credibly fight your entrance) and with an already established installed base (generating reputation and network effects). Hence, despite the weakness of copyright as a tool to protect investments in innovation, "me too" competition (that is, coming to the market as a late comer with a clone of the product of the first comer) does not usually generate market failures in software markets. That simple "functional clones" does not win in software markets was very clear to Microsoft's managers fighting against Netscape, as described by J. Jackson in his findings of fact:

> In late 1996, senior executives within Microsoft, led by James Allchin, began to argue that Microsoft was not binding Internet Explorer tightly enough to Windows and as such was missing an opportunity to maximize the usage of Internet Explorer at Navigator's expense. Allchin first made his case to Paul Maritz in late December 1996. He wrote:
> 'I don't understand how IE is going to win. The current path is simply to copy everything that Netscape does packaging and product wise. Let's [suppose] IE is as good as Navigator/Communicator. Who wins? The one with 80% market share. Maybe being free helps us, but once people are used to a product it is hard to change them. Consider Office. We are more expensive today and we're still winning. My conclusion is that we must leverage Windows more. Treating IE as just an add-on to Windows which is cross-platform [means] losing our biggest advantage — Windows market share.'[115]

Indeed, tying strategies from established platform controllers[116] are precisely what may turn "me too" competition into something feasible (at least if competition authorities allow the adoption of this kind of

[115] *J. Jackson's Findings of fact*, § 166.

[116] E.g. producers of operating systems and other middleware, like browser, or of widespread "killer applications" and maybe, in the future, controllers of very popular portals and suppliers of software services through the Internet (e.g. Google).

strategies). These strategies may generate network effects and overcome barriers to entry in an especially cheap way.

Of course, several strategies of software firms are aimed at generating the highest possible level of direct and indirect network effects for their products. And this is clearly a legitimate strategic goal, dictated by the nature of software goods and benefiting consumers and producers of complementary goods.[117] It is when tying is used as a strategy to generate widespread network effects, benefiting a certain software program, chosen to protect or reinforce a dominant position that competition policy concerns may arise. This has been done, for instance, by Microsoft for its browser, Internet Explorer, and for Windows Media Player. Bundling these programs with Windows, Microsoft had been able to quickly grant them a sufficient number of users, thus making them attractive for producers of content and other complementary goods.[118] Indeed, it has been observed that the old and well-known Chicagoan result, of tying uselessness in traditional markets as a leveraging tool, may be questioned in markets with strong network effects, even if it is true that tying is an effective strategy only for firms with widespread products and significant market power.[119]

What I described may generate two kinds of reactions. One may argue that problems are caused by the fact that copyright is not protecting the real sources of the value of software, so we need software patents and/or a *sui generis* protection to avoid market failures and to make abuses from dominant undertakings more difficult. However, as the first papers of this dissertation showed, the existing system of intellectual property protection of software also presents several advantages, so that this approach may not be particularly advisable. Hence – since copyright in general seems to work quite well as a way to generate innovation in software markets and we may have, at most, a few isolated problems – I suggest looking at alternative solutions. In particular, I submit that it is more sensible to fight specifically predatory tying strategies, with particular care to cases in which platform incumbents are leveraging their position to catch up in complementary markets.

4.1.1. Platform leaders as late comers in new complementary markets

It is a *topos* of software markets that small companies and start-ups are the kind of firms that are introducing new categories of applications and revolutionary approaches, not super-dominant incumbents. Moreover, it is similarly commonplace that incumbents, like Microsoft, will arrive on the scene when an application is already becoming mainstream. At this stage, the incumbent will try to take back control over this piece of software, integrating it into the platform, for instance as a "component" of Windows. Up to a certain extent, I even argue that this may be natural and possibly acceptable. Indeed a way to reach this result may be to buy the producer of the new application and transform it in a division of the platform producer. That, if an application is a must-have for each and every user, will not create inefficiencies. On the contrary, no producer has an incentive to provide this new software service at a cheaper price (or with a better quality) than the

---

[117] It may be more problematic that – in some cases – it is more cost effective to reduce network effects enjoyed by competitors, than to improve network effects of one's own software: the most famous example is the so called "pollution" of Java language/system by Microsoft. To make a long story short (and with some simplifications), Sun Microsystem's Java technology is composed of a programming language and pieces of software called Java Virtual Machines (JVM). A JVM is a software that can be installed on several operating systems and that is frequently used together with a browser. As suggested by its name, a JVM act as a kind of virtual computer, that receives orders through the Java language in an operating system (OS) independent way and takes care of translating these orders in the "language" of the specific OS on which it is installed. In this way, Sun created an additional layer between the OS and the application software, allowing programmers to design multi-platform applications: a feature that is particularly useful for Web-related small applications.

Initially, Microsoft (apparently) cooperated with Sun, acquiring a license to develop a JVM for Windows, which was technically superior to the one produced by Sun (probably also because Microsoft had a better access and knowledge of the workings of Windows OS), and then also produced programming environments for developers using Java (working with Windows, but wanting to produce multi-platform applications, usually Internet related, using Java). Then Microsoft started to introduce new functionalities on "its" version of Java: these functionalities where suggested by Microsoft's programming environment and worked in Microsoft's JVM, but they did not work with other kinds of JVM, running on non-Windows OS. Internal e-mail from Microsoft's employees confirmed that the entire Java strategy of Microsoft was precisely aimed at "polluting" Java, destroying its multi-platform characteristics. (Sun signed a multi-billion $ agreement with Microsoft in order to settle a lawsuit concerning this case, but in the meantime, Microsoft had enough time to create his own competing technology. Today Java is still very used, even if its role as Internet's "Esperanto" has been reduced, and Sun is increasingly considering the possibility of turning it into a fully open source technology.)

[118] Some authors even used the expression "predatory innovation" to describe Microsoft's strategies, in which new products – more or less innovative – where distributed and developed in such a way to increase rival's costs.

[119] See references quoted in *supra* note 9.

platform incumbent, which could internalize the complementarity between the new program and the platform and which already extracts consumers' surplus with the price of the platform itself. In principle, any other solution could be statically inefficient and result in double marginalization. I will discuss more this intuition in § 6.

That said, it may be important to guarantee that the "fights" to "bring back", for instance, to Windows new potential "components" will be as fair and competitive as possible, because it is precisely during this struggle that innovation may speed up and consumers may gain a significant surplus. When software competition becomes stifled, prices drop rapidly to zero (as marginal costs) and software houses race mostly on quality. Indeed, it is also reasonable to suspect that the longer the race, the better for consumers, because – once a player wins[120] – increases in quality are likely to slow down significantly. This is the main reason why, even in cases in which the final victory of the platform controller may be probable (and possible efficient), giving it a way to win faster may be a bad idea.

About the competitive position of the platform owner, notice that, as I will discuss, bundling is not sufficient to impose a certain application on the market: the bundled program also needs to be at least as good as the competing ones, with the only small advantage of the cost of download and installation. Nevertheless, the advantages of being bundled slowly but steadily increase, since users not making any choice implicitly choose the platform controller. Hence, even when third party applications are freely downloadable from the Net, they may progressively loose ground. Indeed, many users are uninformed and/or uninterested in trying "other solutions", once they have a decently working piece of software available on their PC. (In several cases, it is even rational for uninformed users to stay with what they have, given the risk of downloading and installing by mistake some malicious software, a risk that is much higher for inexperienced users.) Thus, bundled software will, in any case, grow some kind of installed base and this will – unavoidably – increase its real value for third parties (both other users, liking direct network effects, and developers, "counting" on the presence of this piece of software on their customers' PCs). Additionally, among the reasons making free downloads a very imperfect substitute for the existence of a program already installed on a PC there are also the limitations forcing business users not to download and/or not to install certain kind of programs. For instance, some firms may not prevent their employees from listening to music while working (after all, it may even increase productivity in open-space offices), however, the majority of firms may prevent workers from installing RealOne (or any other unauthorized software) on their PCs (and this prohibition may be technically and rigidly enforced thought technological measures, limiting the user's possibility of modifying its system). Moreover – despite the fact that the ICT division of a firm may prefer RealOne to WMP – the fact of being unable to uninstall WMP could discourage the installation of RealOne, just to keep systems as "clean" and simple to administer centrally as possible. (The same reasoning applies to OEMs, not wanting to answer calls concerning two different installed programs providing the same function). This is exactly the same reasoning I make when thinking about installing Firefox on my grandmother's PC. "Internet Explorer will be present in any case, and will automatically open from time to time because of some other program's calls: I am not going to put another browser there that may confuse her and confuse me when she calls to say that 'Internet is not working!'".

It should also be observed that what I described above is not necessarily at odds with what incumbents argue about their "being innovative". For instance, Microsoft submits to be investing more in R&D than any other software developer.[121] Certainly, part of this investment will generate genuine innovations (in the patent law sense of the term). However, it is not easy to understand how much of this research is aimed, instead, at creating functional clones of already existing solutions. In other words, it is far from clear whether an investment concerns (using this wording as I did in the second paper) research or development costs. The boundary is admittedly blurred. However, it is possible that some of Microsoft's expenditures are related to the development of new and/or better products, with particular reference to their design and/or specification; while some other expenditure may be related to the activity of individuating the best competitors in specific complementary markets and functionally cloning them. According to the ICE paradigm, this kind of activity is especially likely when these complementors/competitors represent a potential threat to Microsoft's monopoly in operating systems, because in these cases Microsoft has additional incentives to extinguish competition and gain control of other markets. A typical example may be

[120] The winner of the "race" will frequently be the dominant platform owner, unless newcomers create really disrupting innovations. However, it is very possible, as I already hinted, that such a "victory" will involve the buyout of the newcomer.

[121] For instance, US $5 billion in 2001. LE, *Microsoft Europe*, p. 570.

the catching up of Microsoft's Internet Explorer on Netscape's Navigator browser. That case also showed – at least at anecdotal level – that the pace of innovation may slow, once the menace coming from the potential competitor is extinguished. It is the perception of several users (including the author of the paper at hand) that Microsoft's innovation in the browser field had been quite low during the long dominance of Internet Explorer version 6 (period 2001-2005),[122] which followed the end of the "first browsers' war" against Netscape. However, Microsoft's pace of innovation increased once again when Firefox started being a significant competitive threat, bringing to the market various innovations (sometimes borrowed from Opera), such as tabbed browsing and customizable search engines. Indeed, several of these innovation were introduced by "minor" browsers and then "imported" in Internet Explorer version 7, which is now being "imposed" on the market through an automatic process of Windows Update. And Internet Explorer 8 seems to be following with an increasing pace of "innovation".

4.2. Microsoft and the complementarity between tying and information withholding

As I already mentioned, in its Decision of March 2004, the European Commission found that Microsoft had infringed Article 82 of the Treaty by:

> (a) refusing to supply the Interoperability Information[123] and allow its use for the purpose of developing and distributing work group server operating system products, from October 1998 until the date of this Decision;
> (b) making the availability of the Windows Client PC Operating System conditional on the simultaneous acquisition of Windows Media Player from May 1999 until the date of this Decision.[124]

Despite the fact that the Commission unified the investigations concerning the violations mentioned at point (a) and (b) above and issued a unique Decision, Microsoft alleged antitrust violations in the workgroup sever operating systems[125] market and in the media players market are frequently described as completely disjoined:

> Microsoft's misuse of its market power took two forms. […] The only conceptual link between the two courses of conduct was the fact that in both cases Microsoft was attempting to leverage its dominant position in the world market for personal computer operating systems into a different market.[126]

As this paper will show, that is not entirely correct, even though the European Commission did not do much to show any direct link between information-withholding and bundling strategies of Microsoft, apart from describing both of them as leveraging strategies.[127] Actually, the present work stresses that keeping interoperability information secret and practicing strategic bundling are two related strategies, which strongly complement each other, up to the point of both being likely to be necessary in order to extinguish competition on the merits.

In particular, in the European Microsoft case, the tying of Windows Media Player would have been worthless if Microsoft had not been able to keep secret the specification of DRM technologies used by its player. In fact, it was arguably at the control of this market that Microsoft was aiming and because the

---

[122] See the Wikipedia's page devoted to Internet Explorer: http://en.wikipedia.org/wiki/Internet_Explorer.

[123] See *Commission's Microsoft Decision*, Article (1) for definitions. In particular, "the term Interoperability Information means the complete and accurate specifications for all the Protocols implemented in Windows Work Group Server Operating Systems and that are used by Windows Work Group Servers to deliver file and print services and group and user administration services".

[124] *Commission's Microsoft Decision*, Article (2).

[125] Operating systems for workgroup servers are those operating systems "designed and marketed" to deliver "work group server services". These services include three categories of functions: "the sharing of files"; "the sharing of printers"; "the administration of groups and users". The last category can be seen as a kind of meta-service, "in particular ensuring secure access to network resources and the secure use of those resources", hence it comprehends authenticating users and checking users' rights to require a particular task of the server. *See Microsoft CFI*, § 160—162; *Commission's Microsoft Decision*, § 54.

[126] EAGLES & LONGDIN, *Microsoft's Refusal to Disclose*, p. 205—206. Moreover, the authors evaluate that "[o]f the two [conducts], it is only the first that is likely to have resonance for intellectual property lawyers and the only one which is explored [in their paper]."

[127] The Court of First Instance notices (Microsoft CFI, § 1327) that: "It is apparent from recitals 1061 to 1068 [of the Commission's Decision] that the Commission, while recognising the existence of two separate abuses, none the less considered that Microsoft committed a single infringement, namely the application of a strategy consisting in leveraging its dominant position on the client PC operating systems market (see, in particular, recital 1063 to the contested decision)." However, no additional comments are provided about synergies among Microsoft's violations.

availability of free platforms to generate specification-compliant content files would have deprived Microsoft of any possibility of earning a profit on the media player technology it was giving away for free.[128]

Similarly, not disclosing interoperability information concerning Microsoft's workgroup server protocols would have been largely ineffective as a tool to foreclose competition, if Microsoft had not bundled with Windows small applications using the client side of these technologies. It is because of this bundling that users expected these "features of Windows" to work with Microsoft's competitors' servers as well and got frustrated if they did not. It is because of this bundling that network administrators may prefer to use the network features already available in Windows, instead of installing the client-side applications offered by competitors, such as Novell.[129] Overall, it is because of the combination of innovation, tying and information withholding that Microsoft's competitors' products seem unable to fully exploit the technological advances offered by Microsoft's latest client products.

Indeed, secrecy is a key tool to maintain market power and to be able to exploit it, but bundling is a similarly powerful instrument in order to leverage the existing market power in adjacent markets, in which a platform incumbent was not able to establish itself before competitors. I will come back to this subject, but a short explanation of this complementarity may be useful here, along with some comments with respect to the shortcomings (from the point of view of the dominant undertaking) of these strategies when used alone.

Secrecy alone may provide some competitive gains as an exclusionary tool, but it is a very imperfect instrument. If secrecy concerned the already dominated market (e.g. client operating systems), not only an excessive degree of secrecy would be very likely to be sanctioned by antitrust authorities,[130] but also it may turn out to be counterproductive in the market. In fact, platform controllers want to favor interoperability with the majority of software developers, which are increasing the value of their platforms by creating complementary products. It is true that there may be a small minority of developers that worries incumbents because it creates products, which are both complements in the short run and have the potential to pose a competitive threat in future. However, because of the difficulty in discriminating disclosure between "friends" and "foes", recurring too much to secrecy and generally keeping a tight control on a wide range of APIs and CPs may reduce the appeal of a software platform. Indeed, the very success of Microsoft over Apple as the dominant producer of PC operating systems may also be seen as a product of the higher degree of openness of Microsoft's platform with respect to Apple's.[131] Notice that secrecy would be even less effective if it concerned a new product complementary to Windows, e.g. WMP. In this case, Microsoft's product would likely be as good as that of competitors, but with a reduced set of interoperable complementary products (add-ons and similar extensions). Not really a technological boost.

In principle, the best way to use secrecy (without tying) to gain a competitive advantage could be to keep secret just a few powerful APIs used by the dominant incumbent's complementary product (e.g. WMP), but not necessary to (almost) any other piece of software. This, indeed, could offer to the incumbent's complementary product a small "artificial" technical advantage over the competitors' products. Assuming that, ceteris paribus, all the products are of comparable quality, this may be sufficient to conquer the market. However, it is worth noting that such a strategy (e.g. keeping secret a function of Windows just in order to boost WMP) could be discovered by reverse engineering and that it would be difficult for the platform

---

[128] See, for instance, KAI-UWE KÜHN & JOHN VAN REENEN, *Interoperability and Market Foreclosure in the European Microsoft Case.* LSE Centre for Economic Performance (February, 2008), p. 26—27.

[129] Notice that the Commission's finding that "the work group server operating systems market was characterised by the existence of significant structural and behavioural entry barriers" significantly relied on "Microsoft's refusal to disclose interoperability information". *See* EAGLES & LONGDIN, *Microsoft's Refusal to Disclose*, p. 206. I would argue that – absent the tying *and* the withholding of information concerning the client side of Microsoft workgroup technologies – the Commission would not have had elements to find Microsoft dominant in the workgroup operating systems market (assuming that such a market has antitrust relevance, another finding which is depending on Microsoft's withholding & tying strategy). Here, the issue of Microsoft effective dominance of the workgroup server operating systems market is largely collateral and hence negligible; anyhow, I discussed it in FEDERICO MORANDO, *Principi tecnici ed economici per l'analisi del mercato delle piattaforme software: Il Caso Microsoft europeo (Economic and technical elements for the analysis of the market for software platforms: The European Microsoft case)*, 12 Concorrenza e Mercato, 165—241 (2004).

[130] Microsoft itself admitted that if the products of its competitors did not have sufficient access to Windows' APIs and CPs in order to make their own functions work this would be a problem that would merit the attention of competition agencies. See *Microsoft CFI*, § 217: "Microsoft states that it 'has agreed with the Commission from the outset that a competition law issue could potentially arise if competitors were unable to develop server operating systems whose functionality is fully accessible from Windows client [PC] operating systems'".

[131] See, in particular, DAVID S. EVANS, et al., *Invisible Engines — How Software Platforms Drive Innovation and Transform Industries*, (David S. Evans ed., MIT Press First paperback ed. 2008), p. 95—97.

controller to offer any justification for keeping secret the API under examination. In fact, WMP would be a completely distinct program with respect to Windows and Microsoft would be artificially *reducing* the value of Windows as a stand alone product for developers, just in order to advantage its own WMP. That having been said, the main problem of this strategy is simply that it would just offer WMP a limited technical advantage that competitors could compensate with other superior technical features (that, unless a lot of API related information is withheld, but in this case the value of Windows would be reduced more significantly and – overall – repeating this strategy for more complementary products could harm the technical characteristics of Windows as a platform).

Similarly, tying is a strategy that does not work spectacularly on a stand-alone basis. First of all, there is by now a wealth of economic and law & economics literature available regarding bundling strategies, so that strategies able to exclude competitors are also likely to be detected by competition policy authorities. Moreover, in software markets, marginal costs are almost zero and it is quite common to give away products for free, recouping costs from advertising and/or additional services. For tying this has significant implications. To really undercut competitors a dominant undertaking would be required not only to sell the tied product under cost, but also at a negative price, in the sense that an effective bundling strategy may require to sell the bundle platform-plus-complementary-product at a price which is lower than that of the platform alone. Indeed, quite an eye-catching case for competition agencies looking for exclusionary practices. That having been said, if the bundle is the only product available, and even assuming that it costs as much as the platform alone, the platform controller may still, as I already briefly discussed, artificially boost the network effects around its product. However, it is here that the complementarity of secrecy is needed in order to exclude competitors in a really effective way.

Think, for instance, of the market for media players. Microsoft, simply bundling it own WMP to Windows, may have succeeded in forcing RealNetworks and other competitors to give away their own players for free. But they were already doing so (at least for the basic version of their players: adding more functions to WMP would just have add the effect of forcing the way of these functions in the basic version of the competitor's players). In fact, the source of revenue of several producers of media players came either from the sale of content (an aspect I will not focus upon, concentrating my interest on technology markets) or from the sale of the server-side of their technology to content producers and distributors. This server-side of the technology allows for the encoding of multimedia products in a compatible format and streaming or offering as a download these contents to users (obviously, in exchange for a fee and/or gaining from advertising). Now, assume that Microsoft's specifications for its own WMP related technologies (including any CPs governing the client-server streaming technology) were publicly known (and not patent protected); assume also that secrecy had not been used in order to keep Windows' APIs, which WMP may use in order to be faster, lighter or anyhow better than its competitors, exclusive to Microsoft. In that setting, Microsoft would be able to ensure the presence of its own player on users' PC, but competitors would still be able to compete on the lucrative server-side of the technology. Moreover, if Microsoft did not develop the best possible player for this technology, competitors could realize a better player, that several users would still be interested in using (because it could be perfectly compatible with WMP, but technically more performing), possibly for a fee.

To conclude, tying without secrecy could be used to impose a *de facto* standard, thanks to the artificial creation of network effects – and that may be quite worrying, if compared to the "natural" affirmation of a standard in the market. However, such a strategy would not likely completely foreclose competitors. Of course, having defined a standard – even if it is a *de facto* public standard – may present advantages for the undertaking defining it (coming first to the market with a full implementation; having the highest degree of coherence between the standard specification and the rest of one's technological infrastructure, etc.), but these advantages are not likely to extinguish competition.

4.3. The working of technological tying

What I label as "technological tying" are "strong bundling strategies," where the tying and tied goods become a unique product, so that consumers and intermediaries cannot "undo" the bundle prepared by the producer: § 4.3.2 and 4.4 will clarify why, in software industries, technological tying is much more problematic, from a competitive point of view, than simple bundling. In this section, I will describe how technological tying works and, to do that, it may be appropriate to start from the efficiencies it may indeed create.

As one commentator puts it, "Microsoft emphasised several times that bundling is good for consumers, without explaining why bundling is good for its business."[132] Indeed, bundling creates significant advantages for consumers and significant advantages for Microsoft, with some negligible direct costs for consumers (but significant costs for competitors, which – in the medium/long run – could result in a significant reduction of incentives to innovate for them and for Microsoft, hence in potential significant disadvantages for customers). The nice feature of this scenario is that the majority of the advantages for consumers can be achieved, avoiding the majority of detriments for Microsoft's competitors and competition in general.

Microsoft could have bundled someone else's media player with Windows, avoiding the cost of developing its own media player and generating advantages for costumers and for developers wanting to be sure of the availability of multimedia capacities in Windows. But why would Microsoft (or its competitors, like Real Networks) have accepted to give away, for free, products which are costly to develop? According to some commentators,[133] the answer does not necessarily entail an increase in the value of Windows. One reason, as I already hinted, may be that controlling media players allows for a degree of control over the two important sources of income of this industry: advertising and server-side technologies (including the encoding software to generate DRM protected content that is played on these players). Of course, in the media industry, a possible strategy is to make money on the player (like Apple with its iPod), but this may be difficult to do, unless you persuade users that your player is superior (and Apple did that with its iPod). Alternatively, you may try to control some standards, like the one used to encrypt the music (and Apple did also that, not in order to increase the price of the music or of the licenses to producers wanting to use this DRM, but possibly in order to create switching costs for users already owning an iPod, inducing them to buy a new-generation iPod, not to lose access to their iTunes-purchased files collection).

Obviously, even assuming that Microsoft gave away for free its Media Player just to create a market for its multimedia formats, there is nothing wrong in giving away a piece of software in order to create or strengthen a market for a complementary product. For instance, one may make available, for free, a reader software which is creating significant direct network effects, and then sell for a profit the editor/creator for the same file format. Adobe is doing essentially that with its Acrobat Reader software, which is free, differently from the quite costly Acrobat software, which is used to create the PDF files, which may then be distributed and easily and freely accessed through the reader. The difference between Adobe's strategy and that of Microsoft is that Adobe had to convince both "readers" and "writers" of the value of its proposed technological solution. Without a doubt, it would have been easier for Adobe to bundle its Acrobat Reader with Windows and then boast of having a huge and incomparably homogeneous installed base of reliable readers. However, Adobe did not have market power to leverage in order to impose its Acrobat to the market, so it had to compensate with quality. A quite socially desirable kind of compensation. Alternatively, one may try to have its software preinstalled on PC, but without using technological tying. For instance, several customers may actually find a copy of Adobe's Reader installed on their PC, but that is just because Adobe encourages OEMs to install it. And this encouragement could come either with direct payment or because of the fact that quality and usefulness of such a solution push OEMs to install this software on computers in their customer's interest (to increase the value of the PC they are selling thanks to these complementary goods available on it and/or to reduce calls to customer services because "the PC cannot open PDF files").

4.3.1. Dummies and advanced users

Let me assume that the population of PC users is formed by a certain amount (the majority) of "dummies" and by some "advanced users". For our purposes, the difference between dummies and advanced users is that dummies do not like to change their PC configuration and to experiment new software, so – if they already have a media player installed by the original equipment manufacturer – they are unlikely to install another one. Of course, dummies would shift to a dramatically better software if they had a really bad one preinstalled, but I assume that differences between the follower's complement and the functional copy may be significant – at least for advanced users – but not major – and maybe even difficult to perceive for dummies). In contrast, advanced users actually enjoy finding new software. They tend to have a copy of the best program available, especially if it is downloadable for free from the net, so, if all users were advanced, the

[132] LE, *Microsoft Europe*, p. 585.

[133] IAN AYRES & BARRY NALEBUFF, *Going Soft on Microsoft? The EU's Antitrust Case and Remedy*, 2 The Economists' Voice, Article 4 (2005).

technically optimal software would always have the possibility of imposing itself using a zero-price strategy, with the effect of discouraging the entry of technically inferior products.

The real world is not so dissimilar from the previous setting. So, if Microsoft was able to install WMP (and only it)[134] onto the PCs of all dummies, Microsoft would also be able to give its Media Player a significant advantage in term of network effects. This is the case, because I assumed that dummies are the majority of users), so that even a technically inferior MP could conquer the market (enjoying network effects "forced" by Microsoft). But is Microsoft really able to do so? In other words, can Microsoft "impose" a product to users?

My tentative answer is: no, as long as OEMs thinks that the product of RealNetworks (to go on with a real world example) is better than Windows Media Player and they are able to decide to install Real One player only.[135] In fact, OEMs have an incentive to maximize the value of what they sell to consumers and the existence of an (inferior) Microsoft product can decrease their willingness to pay for Real One, but – as long as the price of Real One is the same or slightly higher than the one of WMP – they will simply choose the best software (as advanced users would do). In other words, OEMs have an incentive to work as "consultant" for dummies, *de facto* turning them in advanced users with respect to the initial configuration of their PC. Nevertheless it is important to stress the fact that OEMs – to do their job as "consultants" of users – need to be able to install the best media player only, that is to uninstall WMP (if it is inferior) if it comes bundled with Windows. This condition comes from the fact that – for instance because of efficiency reasons related to call centers – OEMs do not want to install more than one MP on each PC (this may confuse dummy users and make the work of help centers more difficult and costly).[136]

As the US vs. Microsoft antitrust case clearly showed, Microsoft is well aware of the role of OEMs as consultants for users in their software choices and the software house may try to interfere in this process. This is why, according to US antitrust authorities, Microsoft (mis)used his intellectual property rights to force OEMs not to remove the Internet Explorer icon from Windows desktops and not to set other browsers as default applications. The aim of this strategy was precisely to counteract the tendency of OEMs to prefer Netscape's Navigator to Microsoft's Internet Explorer (at the time when Navigator was the established leader of the recently born browser market).

4.4. Modularity as a competition policy principle

Evidently, here the point is not that Microsoft should be forbidden from delivering copies of Windows with WMP pre-installed. What I argue, instead, is that Microsoft should not be allowed to force OEMs (nor users) to keep Microsoft's complementary products (like WMP) installed. And that because, as I already discussed, supporting more than one media player may be uninteresting for OEMs or other users' (implicit) advisors. The Commission's Decision, as confirmed by the Court of First Instance, correctly addressed this issue.[137] If OEMs are free to choose complementary products as they like, dominant undertakings, when they are late to the market, can impose an inferior complementary product if and only if they are allowed to offer it at a negative price. And that negative price cannot be negative just because it includes the cost of installation (ε). In fact, also competitors may offer their product preinstalled (at a price of "–ε"), as long as they are free to make agreements with OEMs (possibly even as a group: Sun, Netscape ad RealNetwork could have produced and freely distributed a CD delivering an installation of Java, Navigator and RealOne). Indeed, Microsoft was a late comer to the media player market and, if it had used simple bundling, Microsoft's strategies would not have had high chances of foreclosing competition; or, at most, they would because of the incapacity of Microsoft's competitors to effectively access to the channel of OEMs in order to pre-install their pieces of software. However, Microsoft did not engage in a simple bundling: it technologically tied its products to Windows, so that the possibility of competing against it to "conquer" the "recommendation" of OEMs was prevented or – at least – significantly hindered.

---

[134] It would probably be enough to be able to set WMP as default program in new PC.

[135] Original Equipment Manufacturers are not only the main hardware sellers, they also decide the default configuration of the PC they sell and they are the first channel of distribution for Microsoft Windows.

[136] This has already been highlighted during the *Microsoft III* case and is confirmed by the findings of the Commission in *Microsoft IV* (see *Commission's Decision* at § 851—852 in particular).

[137] See *Microsoft CFI*, § 1146—1150.

If the leader prices the bundle OS+MP less than the OS alone[138] (or if he is able to prevent OEMs from uninstalling the default Windows Media Player) then he is likely to be able to impose to (dummy) users a product which would otherwise be considered inferior. These are the strategies that I label "strong bundling strategies" or "technological tying".

Of course, these strong bundling strategies need not be detrimental for social welfare in the short run: in fact, typically they are not, because they may increase static efficiency, reducing the total price of the bundle.[139] Indeed, it is relatively easy for a dominant incumbent to show that some of its strategies may even enhance social welfare in the short/medium range. However, as Nalebuff noticed,[140] these efficiency gains frequently "arise from reducing the inefficiency of a monopolist." The point is that

> [a] monopolist can exclude an equally efficient competitor, where the rival has all of the same economies of scale and scope in production. The rival is only missing the ability to reduce its inefficient monopoly pricing. To the extent that exclusionary bundling allows a monopolist to profitably disrupt competition in a large number of adjacent or even unrelated markets, this vastly increases the potential harm caused by a monopoly.[141]

In other words, if efficiencies come from the fact of exploiting less (or more effectively) one's market power, these efficiencies should not be a sufficient excuse for exclusionary practices. For instance, exclusionary bundling could make strategies such as two-part tariffs, price discrimination and some two-sided strategies easier. All of these strategies could create some efficiencies with respect to monopoly in their absence, but is that a good reason to allow the monopolization of other markets, with the exclusion of firms that would be able to compete with any other undertaking, when the leverage of monopoly power is absent?

Probably, no answer will always be correct, but this is a case where some assumption could be made and assuming that competition on the merits is beneficial is quite a safe assumption. Especially because – providing that the platform incumbent is able to realize a product which is as good as that of competitors – it always enjoys slightly higher profits than them, thanks to the internalization of the benefits related to the increased demand for the platform. Hence, in the long run, the platform incumbent will likely monopolize all perfectly complementary markets in which no other player is able to innovate more than it. Preventing exclusionary bundling just makes the race to conquest these markets longer and that is likely to generate, as I already discussed, more innovation and a higher consumer surplus. Moreover, as Ayres and Nalebuff observed:

> While having one standard certainly lowers costs for suppliers and consumers, that doesn't justify allowing a monopolist to lever its market power […] to create a proprietary standard in a complementary market […]. […] Potential costs [of the leveraging of a proprietary standard] include setting the wrong standard and reduced incentives for innovation, along with all the inefficiencies associated with establishing a monopoly in the second market.[142] […]
>
> Microsoft argued that allowing different versions of Windows will require web sites — at great expense — to make their media streams compatible with multiple media players rather than just the de facto WMP standard. But the cost to third parties is not a legal justification for staying a remedy.[143] […]
>
> Economics suggests another reason to ignore these costs: websites will only spend resources if the alternative versions of Windows succeed in the market. If these versions succeed, that is the best evidence that Microsoft abused its power by forcing all Windows customers to take Microsoft's media player.[144]

Overall, leveraging a dominant position in the platform market into a complementary market is likely to be considered an antitrust violation both in EU and in the US. That is the case because this leveraging may be

---

[138] "*Jonathan Zuck, president of the Association for Competitive Technology, a trade group supporting Microsoft in the weeklong hearing, said in an interview […] that the commission 'might now be entertaining the notion that Microsoft might have to bribe consumers to buy an inferior product.'*" ("Microsoft in European Court Says 2004 Ruling Is a Failure", by PAUL MELLER, The New York Times, April 26, 2006). I argue that this is precisely what would be needed to overcome network effects, especially with an inferior product: Microsoft would need to "bribe" OEMs with a lower price of Windows to induce them to accept Windows Media Player as well. Bribing directly users would be useless, because advanced users could easily buy the cheapest version and then uninstall WMP, while dummy users do not directly buy this kind of software, but only pre-configured PC.

[139] But this need not be the case: in some cases even a technically superior product can be thrown out of the market. Notice also that this situation could push the follower to sell his product to the leader, so partially restoring efficiency.

[140] NALEBUFF, *Exclusionary Bundling*.

[141] Id., p. 16.

[142] Id., p. 4.

[143] Id., p. 8.

[144] Id., p. 9.

detrimental in dynamic terms, potentially reducing both product variety and innovation. And because such a leveraging is typically not a strictly necessary condition to create the various efficiencies, which I mentioned above.

For instance, had the Commission done more with WMP – e.g. imposing not only its unbundling from Windows, but also a positive price for the WMP – it could have prevented Microsoft from performing several welfare enhancing strategies.[145] However, had the Commission done less – i.e. not imposing the sale of a version of Windows without WMP preinstalled – it would have missed the point of allowing competition on the merits in the markets for media players. (A competition, which would have frequently been both to convince users to choose one's media player and to convince OEMs to pre-install it on the computers they sold.) Indeed, the only flaw I see in what the Commission did about the bundling of WMP with Windows is that it probably did not make clear enough that unbundling was not an exceptional remedy for an exceptional case, but a competition policy principle, likely to be generalized by future Decisions. If the Commission had been more bold in stressing that, it could have had stronger effects on undertakings' expectations, so that the need of new truly ex post interventions would have been reduced.

Not to give a wrong impression, here I also have to stress that the Commission's unbundling remedy is not a very good tool to restore competition: in order to really do that, once Microsoft had had some years to employ its exclusionary strategies, also a remedy like a must-carry obligation (concerning competitors' medial players) should have been imposed. I will discuss more about that below, but my point here is that the Commission's unbundling "remedy" should actually be seen (not as a remedy, but) first of all as an instance of a general "modularity principle", which competition policy should favor and will favor in the future. The main rationale for such a principle would lie on the fact that a modular environment is likely to be more fertile with respect to unexpected innovations than a closed, proprietary one:

> [The] link between innovation and architecture is the focus of the work of two […] Harvard Business School professors […]. Professors Carliss Baldwin and Kim Clark have demonstrated the importance of modular design in facilitating design evolution and hence industry innovation. In the first volume of an intended two-volume work, they demonstrate the fundamental shift in the design of the computer industry, as IBM increasingly modularized the design of its systems, and as regulators increasingly forced IBM to permit the modules to be provided by others. This change reduced the market value of IBM, but that reduction was overwhelmed by the increase in value in the rest of the industry. As they describe it, a "multiplication and decentralization of design options led to the emergence of a new industry structure for the computer industry," and this in turn radically increased the value of the industry.[146]

At the same time, actually *ex post* remedies – like the ones devised by the European Commission in the Microsoft case – may still be usefully imposed for some years and in order to undo part of the damage to competition, which a violation of the modularity principle may actually have already created, and that because, given the existence of significant network effects and switching costs, competitive advantages gained by tying may be persistent, even after unbundling is imposed.[147]

4.5. The complementarity between tying and information-withholding

Some hints of the complementarity between tying and disclosure has already been recognized in some special cases (and quite implicitly) by some authors, for instance by Ayres and Nalebuff.[148] The authors discussed the possibility of a must carry remedy for WMP in the context of the European Microsoft case and found it not completely persuading, unless coupled with an unbundling remedy. Their observations about must carry are interesting in themselves,[149] however, for my purposes, it is even more interesting that they

---

[145] And would also have been likely excessive, according to the exclusionary bundling test, as understood by Id..

[146] L. LESSIG, *The future of ideas: the fate of the commons in a connected world*, Random house, 2001, XIII+352, 92, quoting Y. CARLISS BALDWIN & KIM B. CLARK, *Design Rules*, vol. 1 (Cambridge, Mass.: MIT Press, 2000), 63.

[147] See, among many, IAN AYRES & BARRY NALEBUFF, *Going Soft on Microsoft? The EU's Antitrust Case and Remedy*, 2 The Economists' Voice, Article 4 (2005), p. 4—6: "The Commission picked unbundling over must carry. If they had to choose, they chose correctly. But there was no need to pick one over the other. They could have done both."

[148] Id..

[149] Id., p. 5: "To our mind, the real problem with must carry is that it doesn't go far enough on its own. At first glance, it might appear that with multiple players preinstalled, the result would be level playing field. WMP would be just one on a menu of options from which all consumers would choose, not an automatic "default." Not quite. The other options would vary, and WMP would be a constant. Thus, must carry alone would not have solved the underlying problem: Microsoft would still have the unique ability to ensure that its media player would be on all new machines — and thus eventually on all machines. That, in turn, would mean that a

concluded that, under a simple must carry obligation (without unbundling), "[r]ival players may hang on, but only if Microsoft deigns to license its decoder."[150] Hence, the authors thought that a disclosure of the Windows Media file format could have been a solution to restore some competition, even in a market in which Microsoft would have been able to impose its Windows Media Player to 100% of PC users, strategically using bundling. In this sense, Ayres and Nalebuff implicitly recognized that tying, without information withholding, is much less effective than the two coupled strategies. In other words, the authors understood that controlling information about the specifications of Windows Media Players' file and DRM formats was an implicit, but necessary condition to control the media player market. Or, said differently, Microsoft's tying was able to extinguish competition only because it was coupled with trade secret on the specifications of Microsoft's DRM and file formats. This is coherent with what I already hinted: if specifications (including file formats and specifications of digital rights management techniques) are not kept secret, bundling may impose a *de facto* standard on the market. However, in the first paper of this dissertation, I already showed that intellectual property (without secrecy) offers a very imperfect control over the reimplementation of a known specification. Thus, competition *within* the *de facto* standard is still possible, unless this standard is kept secret.

4.5.1. Bundling and low prices are not sufficient (better: not sophisticated enough)

Are information-withholding and technological tying strategies necessary, after all? If its products are technically at least as good as those of competitors, Microsoft should be able to conquer the market for server operating systems just by lowering its prices for server operating systems. It may even finance these discounts with the profits generated from Windows client systems, but it would be quite difficult to show that these strategies are predatory, given the zero marginal cost of each system sold (and the advantages of high volume sellers in showing that also average costs are, or will likely be, covered). All that is probably true, however, there are good reasons to adopt subtler strategies.[151] For instance, Nalebuff clearly pointed to the advantages of bundling over simple predatory prices (or even strong price competition).[152] There are indeed multiple advantages, some of which are not so relevant here.[153] One which is relevant is that "[t]he anticompetitive effect of the bundling is […] magnified when there is a possibility of substitution between [the tying and tied goods]."[154] Considering the fact that potential competition is indeed one of the most important exception to the ICE paradigm, and hence one of the most importance motives of exclusionary strategies performed by platform controllers, it may be quite important to remember that – when potential competition is at place – exclusionary bundling works much better than pure price competition (predatory prices included).

To understand why tying may be more effective than simple price competition, think about client operating systems and operating systems for workgroup servers. Normally, each user will run a client operating system and a central computer will run the server OS, even if it is perfectly possible that this same computer is also used by a user as a normal personal computer, especially if the workgroup is very small (think about an office with three or four users sharing files and a printer). On average, it may not be a good idea to let a user use the server also as a normal client, for security reasons; moreover, the substitution between clients and servers for the other users is not convenient, simply because the license for the server OS costs much more than a client OS license. However, assume that Microsoft decided to displace Linux servers from small offices. To do that using just price competition, Microsoft would have to price Windows servers for small workgroups so low as to compete with ordinary Windows clients. Given the fact that Linux servers are even available for free, for small firms not interested in specific installation and update assistance (and/or managing these functions internally), Microsoft would even need to price so low as to undercut Windows clients. In other words, Microsoft wants to compete with server operating systems, however, Microsoft does not want to undercut server systems in an excessively explicit way, otherwise firms could think about buying a server OS instead of

content provider that encoded its content in the WMP format would be ensured nearly 100% reach in the market. And that content provider would have little incentive to engage in "dual encoding" — that is, encoding in a number of different formats."

[150] Id., p. 6.

[151] NALEBUFF, *Exclusionary Bundling*.

[152] Id., p. 24.

[153] Id., p. 1: "The intuition is that under predation, the firm has to actually charge a price below cost and thus loses money that it later has to recoup. Under exclusionary bundling, the firm only has to threaten to raise its unbundled prices if the bundle is not bought. All customers are led to buy the bundle and so the threat never need be carried out."

[154] Id., p. 23.

a client one.[155] Thus, Microsoft may prefer subtler strategies, discriminating competing servers in complex ways through bundling and information withholding, instead of simple price competition threatening its own Windows client profits.

4.6. Microsoft workgroup-servers-related violation as a tying

How useful some forms of tying are in order to gain advantages from information-withholding may be understood describing the workgroup-servers related part of the European Microsoft case. Typically, this part of the European Microsoft case is depicted as a refusal-to-deal/information-withholding issue. However, tying with the dominant Windows operating system was at play here as well. Indeed, the Commission and the Court noted that

> the changes brought about by the move from Windows NT technology to Windows 2000 technology and Active Directory include the fact that a number of functions are integrated both in the Windows 2000 Professional operating system and in the Windows 2000 Server operating system, in order to simplify the administration of Windows client PCs in Windows domains.[156]

Of course, this simplification is achievable only by Microsoft, since it (also) derives from the fact that it is no more necessary to separately install and update the client-side part of Microsoft's workgroup technologies. A specific example concerns distributed files systems, which are used by operating systems in order to provide transparency with respect to the actual physical location of files and redundancy, in order to insure data availability and/or robustness in case of disasters.[157] In this field,

> at the end of the 1990s Microsoft marketed such a system, called 'Dfs' (Distributed File System), in the form of an add-on that could be installed on client and server PCs running Windows NT 4.0. Windows 2000 is the first generation of Microsoft products to include native support for Dfs both on the client PC and the work group server side.[158]

Once again, not all of the features of this technology are kept strictly secret, however the most advanced ones (providing fault tolerance and increased reliability) are.[159]

Overall, it is difficult to deny that tying was part of Microsoft's strategy to conquer the workgroup server operating systems market. Clearly, such a tying would just have amounted to an improvement of Windows clients' workgroup capabilities, absent information-withholding. At the same time, information withholding, without tying, would not have allowed Microsoft to leverage its crucial advantage: the super-dominance in the client operating systems market.

4.6.1. Interoperability with windows clients is the key for the server market

That interoperability with Windows client operating systems (in particular with the Professional version) and with Office (again, in the version addressed at businesses) has been the core of Microsoft's strategy to conquer the workgroup server operating systems has been observed also by European Competition authorities. Indeed, there are reasons to believe that this was the main (and possibly the unique) competitive advantage of Microsoft. To ascertain that, the Commission ordered several market surveys.[160] These surveys showed that a key factor in determining purchase decisions concerning workgroup server operating systems was the "interoperability with Windows work stations (average mark of 4.25 [out of 5])", accompanied by the "reliability/availability of the server operating system (average mark 4.47)" and "integrated security in the server operating system (average mark 4.04)". Indeed, in the highest ranked of these criteria, Windows obtained the lowest mark (3.63), surpassed by UNIX (4.55), Linux (4.10) and NetWare (4.01) workgroup products. Similarly, Windows obtained the lowest average mark for the security-related criterion (3.14),

---

[155] See also KÜHN & REENEN, *Interoperability and Market Foreclosure*, pp. 18—19.

[156] *Microsoft CFI*, § 171. See also §§ 155—157 of the *Commission's Microsoft Decision*.

[157] For more technical information, see the Wikipedia related voices: http://en.wikipedia.org/wiki/Distributed_file_system and http://en.wikipedia.org/wiki/Distributed_File_System_(Microsoft). See also *Commission's Microsoft Decision*, §§ 161—163.

[158] *Microsoft CFI*, § 173.

[159] See, for instance, *Commission's Microsoft Decision*, § 164: "Under Windows 2000, Dfs can be set in two different modes: 'stand-alone' or 'domain-based'. The domain-based mode, which provides a number of advantages in terms of 'intelligent' retrieval of the Dfs information from the client PC, is available only in Windows domains, and enhanced by the presence of domain controllers running Active Directory.".

[160] See *Microsoft CFI*, § 397—411.

surpassed by UNIX (4.09), NetWare (3.82) and Linux (3.73). It is only in the field of "interoperability with Windows work stations" that Windows excelled, (average mark 4.87), followed by NetWare (3.78), Linux (3.43) and UNIX (3.29).

Since the interoperability of non-Microsoft's workgroup servers with Windows has not been reduced by Microsoft's actions, the core of Microsoft's competitive advantage has to be looked at in the new functions, which the software house decided to tie with Windows Professional clients, starting with Windows 2000. Of course, it is possible (and even likely, in my opinion) that these functions and Microsoft's workgroup servers really generated significant improvements with respect to past workgroup computing experiences. However, it is far from clear whether there was any technological need to tie the client-side of these new technologies to Windows. Indeed, Windows (as basically any complex software product) is designed as a modular system, as the multitude of available versions of it shows: Windows Professional itself is just a version of Windows with added modules. Technically, Microsoft could have disclosed the APIs used by these modules to integrate with Windows (and some of these APIs were likely already public) and – what is more important – it could also have allowed OEMs and network administrators to remove these modules and substitute them with the modules produced by other workgroup server developers. The fact that Microsoft, once again, did not follow a modular strategy makes it difficult to discern whether Microsoft simply realized better products or the tying of the new functions with Windows granted Microsoft's server an "artificial" competitive advantage. For sure, Microsoft workgroup products ended up being the only ones "fully compatible" with a standard installation of Windows Professional and that without any fault on the part of Microsoft's competitors.

As the Commission recognized (and the Court of First Instance confirmed), it is "in view of Microsoft's quasi-monopoly on the client PC operating systems market [that] its competitors are not in a position to develop viable alternatives to its communication protocols."[161] Hence, the strategic leverage of Microsoft's monopoly power had as a fulcrum its Windows (client) monopoly (and it could not have been otherwise). Moreover, given the quasi monopoly of Microsoft's Office suite on business personal computers, also the Office monopoly contributed to this strategy.[162] It is possible that Microsoft could have gained a dominant position in the market for workgroup server operating systems[163] even without the help of tying. However, it is my opinion that also in this market the effectiveness of the leveraging strategy has been maximized by a combination of information withholding and tying. Indeed, I submit that the speed at which Microsoft has been able to impose its solutions on the workgroup servers market required the tying of the client-side of Microsoft's server technology to Windows (client) as a necessary condition; otherwise, Microsoft workgroup protocols would have likely been as good as any other, but with the additional "defect" of secrecy and reduced interoperability in an environment mixing servers of various producers. Again, in this context, secrecy may make business sense in the short run only in two cases. If, as Microsoft did, tying is performed and secrecy concerns the communication protocols used by the client-side of the technology to communicate with the server-side; or – if tying is not performed for some reasons – if secrecy concerns some APIs of Windows, used by the client-side technology, which is normally distributed along with servers' operating systems (as Microsoft's competitors are forced to do). To be sure, in the long run, secrecy could have allowed Microsoft to keep better control of its own workgroup server solution. However, I doubt that secrecy alone could have helped Microsoft in quickly gaining control of the workgroup server industry. Instead, coupling of secrecy and tying (surely assisted by attention to the needs of customers, to ergonomics and to the general design of its technological solution) allowed Microsoft to quickly achieve this result.

Summarizing, Microsoft abuse – if any – must be related to the client market, in which Microsoft is super-dominant. This abuse will ideally consist of a tying and of an information withholding strategy. As I already stressed, should tying not be carried out, Microsoft looses part of its advantage, but it may still disadvantage competitors if it is able to use in its own complementary products some APIs of Windows, which are unavailable to the competitors. However, unless secrecy concerns a very significant amount of information, information withholding alone is much less likely to be sufficient to leverage market power that the equivalent strategy coupled with tying. That is the case, because the super-dominant undertaking's advantage revolves only on a more or less significant technological boost and cannot take advantage of strategically created network effects and switching costs (which would be related to the fact that Microsoft's client-side technology is already available on Windows clients, when tying is performed). In fact, another reason which

[161] *Microsoft CFI*, § 684.

[162] About the strategic importance of Microsoft Office suite, see also J. Jackson's findings of fact, § 344 ff..

[163] See *Commission's Microsoft Decision*, §§ 473 ff.

makes major information withholding unlikely is that this would be very costly for Microsoft, in terms of missed opportunities of creating value for all the producers of complementary products: indeed, several of the same APIs of Windows, which could be useful for Microsoft's potential competitors, could be appreciated also by other developers building Microsoft's "application barrier to entry".

Moreover, withholding information concerning Windows APIs, which are used by some Microsoft's middleware and/or client-side network technologies (which can be described as middleware exposing CPs, instead of APIs), would be a very difficult strategy to justify in front of competition authorities. At present, such a strategy would also risk violating Microsoft's US Consent Decree[164], which provides that

> Microsoft shall disclose […] for the sole purpose of interoperating with a Windows Operating System Product […] the APIs and related Documentation that are used by Microsoft Middleware to interoperate with a Windows Operating System Product.[165]

Indeed, the US Consent Decree, even though poorly enforced, also provided that

> Microsoft shall make available for use by third parties, for the sole purpose of interoperating or communicating with a Windows Operating System Product, on reasonable and non-discriminatory terms […], any Communications Protocol that is, on or after the date this Final Judgment is submitted to the Court, (i) implemented in a Windows Operating System Product installed on a client computer, and (ii) used to interoperate, or communicate, natively (i.e., without the addition of software code to the client operating system product) with a Microsoft server operating system product.[166]

Hence, to make these provisions coherent with my recommendation, the only necessary addition (mainly for sake of clarity) could have been a clear statement that any Microsoft's client-side program embedding client-to-server communication protocols must be considered a "Microsoft Middleware" (and/or part of a Windows Operating System Product) for the purposes of previous provisions.[167]

4.6.2. Was the US consent decree already sufficient to prevent Microsoft's violations?

It is my opinion that the Commission should have tried to convince Microsoft to sign an agreement similar to the US Consent Decree also in Europe, in order to be able to timely and effectively enforce these principles in Europe as well. Indeed, that could have had the effect of creating a more proactive and efficient enforcing of these principles. (It has been observed that, in the US, such an enforcement had not been particularly effective. Possibly, also because of the fact that the Bush administration had a clearly different opinion of Microsoft's behavior, with respect to the Clinton administration, under which the Department of Justice had started the US Microsoft investigation, which lead to the Consent Decree). One can also submit that understanding the complementarity between tying and information withholding may suggested that competition policies adopted toward Microsoft on both sides of the Atlantic are far more compatible than argued by the press, part of the literature and government officials. Also compare what has been decided about Internet Explorer (IE) in the US and about Windows Media Player (WMP) in the EU. On one side of the Atlantic, the Courts and the DoJ tried to prevent Microsoft abuses imposing disclosure obligations concerning APIs and similar interfaces used or exposed by Microsoft's middleware (including IE): admittedly, they did not save Netscape from disappearing, but created an environment where Mozilla's Firefox has been able to reach a market share of about 20 percent. Instead, on our side of the Atlantic, the Commission tried to impose modularity, favoring competition between different media players – and between different media formats – instead of allowing competitors to fight "inside" Microsoft's *de facto* standard. Consider also that differences concerning the amount of fines – where the Commission seems to have been exemplar, while nothing happened in the US – are misleading. It is true that the Commission inflicted a huge fine on Microsoft, but one should be aware that – in the US – Microsoft escaped paying very high damages (likely triple punitive damages) to the owner of Netscape's Navigator just because – in the mean time – the company

---

[164] State of New York, et al. v. Microsoft Corp., U.S. District Court for the District of Columbia, Civil Action No. 98-1233 (CKK), Final Judgment of November 1, 2002 (hereinafter: *US Consent Decree*).

[165] *US Consent Decree*, sect. III.D.

[166] *US Consent Decree*, sect. III.E.

[167] In the *US Consent Decree*, sect. VI.K, the term Microsoft Middleware is already defined in quite a broad way, explicitly encompassing the majority of technologies tied to Windows at the time (), but it may be disputable that this definition also encompasses the client-side of any network technology (which is not tied to Windows clients: in fact, tied client-server technologies were already covered by section III.E).

had been bought by AOL. Hence, instead of litigating, Microsoft decided to reached a multi-millions $ agreement with AOL – likely containing clauses concerning a settlement of any private antitrust enforcement procedures – just because AOL was in a period of financial distress. In fact, also after the famous European fine of more than 400 mln €, it is on the other side of the Atlantic that a very high price (of more than 600 mln $) had to be paid by Microsoft, in order to settle a potential litigation with Sun Microsystems (again, a "strategic agreement" looking much more similar to an antitrust "private" fine).

All that having been said, also the US Consent Decree was a far from perfect solution to Microsoft's abuses.

First of all, the US disclosure obligations, even if appropriate in principle, were a temporary remedy and had been established as an agreement and not as the effect of any clearly individuated violation performed by Microsoft. Hence, the Consent Decree did not create a clear precedent for Microsoft and, in particular, it did not create any precedent for other super-dominant undertakings. Instead, as Microsoft observed, not without some understandable worry, the Commission's Decision has implications "extend[ing] indefinitely into the future":[168]

> it is clear from […] the contested decision that the Commission considers that the obligation to disclose interoperability information must apply 'in a prospective manner' to future generations of Microsoft's products.[169]

Indeed, comparing it to the US Consent Decree, this is likely the most appreciable aspect of the Decision. In fact, I do not see any reasons for which Microsoft's Windows-client-related disclosure obligations should last any less than its super-dominance of the client PC operating systems market, precisely because the very reasons for these obligations is – above any specific behavior of Microsoft – the "special responsibility" associated with its super-dominance. To the contrary, it is clear (also) from the arguments, which Microsoft used in front of the European Court of First Instance, that the software house expected to gain back "full control" of its trade secrets after five years from the US Consent Decree.[170] Probably, exclusive control will indeed be granted back to Microsoft, apart from an extension of these obligations, due to the still imperfect implementation of some of the Consent Decree provisions.[171]

Moreover, also the US Consent Decree did not fully take into account the complementarity between tying and information withholding, which I described in this paper. Assuming that, from the point of view of US competition authorities, disclosure is to be preferred to unbundling (possibly because of the efficiencies, partly shared by consumers, coming from tying), if Microsoft decided to use technological tying with its super-dominant platform, this paper would suggest requiring the software house to meet the terms of a very extensive disclosure obligation. Such an obligation would encompass, for instance, the specification of any file format used by applications tied to Windows, including – among others – the specifications of the digital rights management (DRM) technologies used, for instance, by its Windows Media Player. However, there is no hint of similar obligations in the US settlement. Hence, it is evident that the US Consent Decree was very far from enforcing the suggested level of disclosure as an alternative to unbundling. On top of that, the US "remedy" was both temporarily limited and backward looking in identifying the kind of information that Microsoft was obliged to disclose.

Finally, it is important to notice that – concerning the workgroup server market – the Commission also wanted to tackle the issue of server-to-sever interoperability, since

> Under the [US Consent Decree], [Microsoft] is required to license communication protocols implemented in Windows client PC operating systems for the sole purpose of being implemented in server software.

---

[168] *Microsoft CFI*, § 674.

[169] Microsoft CFI, § 1268—1270. "The importance of interoperability with planned future purchases means that the disclosure order should apply in a prospective manner to future generations of Microsoft products. Accordingly, the disclosed information will have to be updated each time Microsoft intends to bring to market new versions of its relevant products." See also the *Commission's Microsoft Decision*, § 1002.

[170] *Microsoft CFI*, § 673: "The applicant's obligations under the United States settlement are limited to a five-year period […] and an undertaking has a greater incentive to continue to develop technology when, after a fixed period, it will again have exclusive use of the improvements to that technology.".

[171] In May 2006 Microsoft agreed with the US DoJ to extend selected provisions of the Decree until November 2009. The last formal extension came on January 29, 2008, when Judge Colleen Kollar-Kotelly extended the consent decree through November 12, 2009. (For further information and links, see http://www.microsoft.com/presspass/presskits/consentdecree/.) It is interesting to notice that the Judge specified that the "extension should not be viewed as a sanction against Microsoft." However, the Court also had to admit that "the fact remains that more than five years after the Final Judgments were entered, the technical documentation required by Section III.E is still not available to licensees in a certifiably complete, accurate, and useable form."

> Under the [Commission's Microsoft Decision], on the other hand, it is required to license its 'server/server' communication protocols so that they can be implemented in directly competing server operating systems.[172]

This is a complex issue, requiring specific attention.

4.6.3. Server-to-server interoperability and the broadness of interoperability in general

In its Decision,

> [t]he Commission emphasises that it is important […] that the interconnection and interaction involving the Windows 2000 Professional source code should not be viewed solely as being intended to enable a particular Windows work group server to communicate with a particular Windows client PC. It is more accurate to describe that interconnection and that interaction in terms of interoperability within a computer system encompassing several Windows client PCs and several Windows work group servers, all linked together in a network. Interoperability within such a computer system thus has two inseparable components, namely client/server interoperability and server/server interoperability […].[173]

However, a first point to clarify is that dominance in the workgroup server market was not a necessary element for the Commission's Decision. Indeed, some commentators expressed doubts as to the soundness of the Commission's relevant market analysis concerning the workgroup server operating systems market, both because of the treatment of potential entry and because of the definition of market shares.[174] However, as the Court of First Instance stressed, Microsoft conducts could have been deemed illicit just on the basis of its (super-)dominance of client PC operating systems:

> Microsoft's abusive conduct has its origin in its dominant position on the [PC operating systems] product market […]. Even if the Commission were wrongly [sic] to have considered that Microsoft was in a dominant position on the [work group server operating systems] market […] that could not therefore of itself suffice to support a finding that the Commission was wrong to conclude that there had been an abuse of a dominant position by Microsoft.[175]

That having been said, I argue that the Commission has been wise in discussing this issue, considering the findings of the US Court of Appeals in the Microsoft III case, were the issue concerning the monopolization of the browser market had been partially remanded, precisely because of the absence of a sound analysis of that market as a separate relevant antitrust market.

My perplexities on the Commission's analysis concerning server-to-server interoperability comes from the fact that – given the double finding of dominance for Microsoft, both in the client PC and workgroup server operating system markets – it is not perfectly clear whether the disclosure obligation concerning Microsoft's server-to-server communication protocols is based on the client super-dominance, on the server dominance, or on both. For the specific case at hand, this is not a dramatic problem. Banalizing the Court's position, the Commission had more than enough arguments to impose a disclosure obligation on Microsoft. However, some more clarity could have been useful, especially in order to create a sound "precedent". Indeed, Microsoft's dominance in the server operating system market is (was) still be far from super-dominance. For instance, its market share amounted to 65 percent, according to the Commission, but only narrowing the definition of the market to "workgroup server operating systems", arguably a market which is defined more from the need to interoperate seamlessly with Windows client PC than because of any other structural difference with higher-end servers. In what follows, I will suggest to condition obligations to disclose to the existence of a super-dominance position, precisely in order to increase the administrability of this obligation and to avoid the possibility that minor, but still very active and dangerous, competitors may decide to wait and see whether the (quasi-)dominant player will be forced to disclose its own technological design.

Indeed, I think that there would have been possibilities to restore competition in the workgroup server market just focusing on disclosure obligations generated by Microsoft super-dominance in the client market. In fact,

---

[172] *Microsoft CFI*, § 673.

[173] *Microsoft CFI*, § 182. See also the *Commission's Microsoft Decision*, § 178.

[174] For a strong critique of the *Commission's Decision*, see ROBERTO PARDOLESI & ANDREA RENDA, *The European Commission's Case Against Microsoft: Fool Monti Kills Bill?*, LE Lab Working Paper No. AT-07-04 (August, 2004). I also proposed some comments about the Commission's analysis of the workgroup server operating systems market in MORANDO, *Il Caso Microsoft europeo*.

[175] *Microsoft CFI*, § 559.

> [i]n many cases […] there is 'symmetry between server/server and client/server interconnection and interaction' […]. The Commission mentions, by way of example, the fact that the same 'application program interface' (API), namely 'ADSI' (Active Directory Service Interface), is implemented both on Windows 2000 Professional and Windows 2000 Server to handle access to Active Directory domain controllers. A further example given by the Commission is the fact that, in a Windows domain, the Kerberos protocol, as extended by Microsoft, is used for authentication both between a Windows client PC and a Windows work group server and between several Windows work group servers. In certain circumstances, 'servers will query other servers on behalf of a client PC' […]. By way of example, the Commission mentions, in particular, 'Kerberos delegation', a functionality present in the Windows 2000 Server operating system which allows a server to borrow the identity of a client PC and to request a service from another server on behalf of that client PC. Thus servers quite frequently address requests to other servers and therefore act as client PCs […].[176]

Hence, even if the Commission had just imposed a disclosure obligation on Microsoft, concerning communication protocols and APIs implemented and tied with Windows clients, competitors could have likely reconstructed the majority of the information needed to also realize server-to-server interoperability (and, what is more relevant, they would have been allowed to receive more help in competing with Microsoft in these fields, in which the client monopoly offered to Microsoft an unbeatable competitive advantage).[177] In other words, the fact that client/server and server/server "interconnections and interactions are closely interlinked"[178] does not necessarily imply that also server-to-server protocols must be disclosed. Indeed, this same fact also implies that a complete specification of the client side of the technology, over which Microsoft exercises complete control and where it performed a technological tying to Windows, could be useful for competitors in order to realize their own server-to-server protocols.

That having been said, I admit that the approach suggested in the paper at hand (i.e. limiting the mandatory disclosure to the client-side of the technology) could fail to ensure a complete server-to-server interoperability. This will certainly be the case, unless (or until) Microsoft is super-dominant also in this market. However, as I am going to show, this happens for good reasons, which could be better understood starting from the debate about the concept of interoperability that divides Microsoft and the European Commission.

Summarizing the discussion, on the one hand Microsoft argues that a sufficient degree of interoperability is achieved whenever a competitor's workgroup server operating system is able to have all its own functions fully working when connected with a Windows client.[179] (Of course, this may need additional pieces of software to be installed on Windows, if the competitor implements new communication protocols; however, this is possible thanks to a plethora of well-known Windows APIs, which could be used to implement these new protocols also on the client side of the system.) Applying the Commission's position in a broad way, on the other hand, one may argue that an appropriate degree of interoperability is reached when a Microsoft's workgroup server operating system can be substituted by competitor's, without any loss of functionality for the entire system. In other words, "all the functionalities of both products must function correctly".[180] (It is

---

[176] *Microsoft CFI*, § 183.

[177] I do not want to be excessively tedious with technical details. However, it may be interesting to notice that the Court observed also (§ 185) that "[s]ome client/server communications build on the expectation that certain server/server communications have already taken place. In particular, when a client PC running Windows 2000 Professional queries the domain controller in a Windows 2000 domain, the client PC will expect 'some preparatory coordination to have taken place between the domain controllers running Windows 2000 Server' […]." I argue that this, instead of being an additional element in favour of forcing Microsoft to disclose server-to-server protocols, is just and additional hint of the fact that stating what a Windows 2000 client expects to receive from a Windows server is sufficient in order to reimplement an interoperable workgroup server solution.

[178] *Microsoft CFI*, § 187.

[179] See also *Microsoft CFI*, § 215: "in its response of 17 November 2000 to the first statement of objections, Microsoft states that the degree of interoperability apparently required by the Commission is not consistent with Community law and does not exist on the market. […] [T]he applicant submits that 'full interoperability is available to a developer of server operating systems when all of the functionality of his program can be accessed from a Windows client operating system' […]." See also § 217: "Microsoft states that it 'has agreed with the Commission from the outset that a competition law issue could potentially arise if competitors were unable to develop server operating systems whose functionality is fully accessible from Windows client [PC] operating systems'".

[180] See also *Microsoft CFI*, § 215: "The applicant [i.e. Microsoft] maintains that the Commission wrongly defines interoperability much more broadly when it considers that, for there to be interoperability between two software products, all the functionalities of both products must function correctly. That, in Microsoft's contention, is tantamount to requiring 'plug-replaceability' or 'cloning'". About "plug-replaceability" and this debate in general, see also R. J. HART, *Interoperability Information and the Microsoft Decision*, 28 European Intellectual Property Review, 361—365 (2006).

evident that this will only happen if the competitor's server OS is appropriately developed, taking into account the technical specifications disclosed by Microsoft.)

In my opinion, Microsoft's "one-way interpretation"[181] of the concept of interoperability must be rejected.[182] However, even though the Commission's interpretation of the concept of interoperability is correct (i.e. it is coherent with the one envisaged by the Software Directive 91/250), it is not clear to me that any dominant undertaking has a duty to guarantee such a full level of interoperability.[183] Indeed, the Software Directive described the degree of interoperability that competitors should be able to achieve through reverse engineering, i.e. through self-help, not the degree of interoperability than an undertaking must willingly guarantee. In other words, the Software Directive described the degree of interoperability that is legitimate to aim at (from the point of view of competitors) and not the minimum degree of interoperability that must be granted (by incumbents). That these two degrees of interoperability need not to be equal is testified by the fact that nobody would waste his own resources on reverse engineering a piece of software which is not already widespread in the market (and likely dominant or quasi-dominant – whatever this may mean – even if not necessarily super-dominant). Hence, what would be the point of a right to perform reverse engineering, if any undertaking also had a right to require interoperability information from its most successful competitors? Indeed, also the Court of First Instance appropriately stressed that:

> The question in the present case is not so much whether the concept of interoperability in the contested decision is consistent with the concept envisaged in [Directive 91/250] as whether the Commission correctly determined the degree of interoperability that should be attainable in the light of the objectives of Article 82 EC.[184]

It is in this context that I argue that a duty to disclose (or, at least, to deal) for a super-dominant undertaking, restrictively interpreted under the essential facility doctrine, may be appropriate, but expanding this duty much further would be dangerous for incentives to innovate.

The approach I propose in this paper, in effect, achieves a result that is midway between the concepts of interoperability proposed by Microsoft and by the Commission. It guarantees the kind of interoperability described by Microsoft and – additionally – it guarantees that a standard installation of the Windows client operating system is able to have all its own functions fully working with both a Microsoft workgroup server operating system and with those of competitors. And that because Microsoft is super-dominant only in the client operating systems relevant market. In other words, my approach mandates full interoperability only with super-dominant pieces of software and other pieces of software tied to them (including the specification of their file formats, if any[185]).

To better discuss this point, it is necessary to clarify some assumptions. In the real world it is possible that Microsoft became dominant or even super-dominant in the market for workgroup server operating systems also thanks to its tying strategies. If that is the case, a disclosure obligation may be appropriate under the approach suggested in this paper (see below, § 4.7). However, let me assume that Microsoft is not super-dominant in this field or – even more simply – let me take the point of view of the launch of Microsoft's Windows 2000 (and the related Active Directory service), when Microsoft was surely not yet super-dominant. As Microsoft observed, and the Commission and Court of First Instance noted, "the directory service is a key competitive feature responsible to a large extent for the success of particular products" and Microsoft's "Active Directory is […] at the heart of Windows server operating systems", also because "[f]or both file and print services and user and group administration services, it [is] important to know with precision which user [is] entitled to access which network resources". [186] At the moment of launching Windows 2000, if Microsoft

---

[181] See *Microsoft CFI*, § 218.

[182] For a full discussion of the reasons of this rejection, see the *Commission's Microsoft Decision*, §§ 750—763.

[183] In fact, also the Commission formally clarifies (*Microsoft Decision*, § 763) that: "contrary to Microsoft's contention, this case does relate to a situation 'where a dominant supplier refuses to make information available which is necessary for interoperability as defined in [the Software Directive]'. *This does not mean that Microsoft's behaviour is automatically abusive.* However, the fact that the Software Directive explicitly mentions the possibility that a refusal to supply information necessary for interoperability may constitute an abuse of a dominant position is not inconsequential for this analysis pursuant to Article 82." (Emphasis added.)

[184] *Microsoft CFI*, § 227.

[185] This specification is not relevant for the workgroup server operating systems, but it is for the tying of media players and similar applications. The concept of "file format" should be interpreted in a broad way. To provide a hypothetical example, if Microsoft decided to bundle an anti-virus software with Windows, it could do so (even if, for technical reasons of incompatibility, making it impossible to uninstall this software would be an abuse), but it would be forced to disclose the format of its virus definition files, so that third parties would be allowed to compete with Microsoft in providing paid update services.

[186] *Microsoft CFI*, § 190.

bundles the client side of its Active Directory technology with Windows and/or if it withholds the information (APIs) used by client-side add-ons in order to interoperate with Windows clients, then – according to my paper – there is an antitrust violation. However, if Microsoft discloses all the APIs of the Windows client OS used by its new technology and does not bundle with Windows clients any new client-side application, what is wrong in Microsoft strategy?

Without tying and information-withholding concerning Windows clients' APIs (where Microsoft is super-dominant), Microsoft, Sun, Novel and other competitors are on a fair competitive ground. They can all develop their preferred workgroup server technology and can all realize client-side pieces of software designed to "expand" Windows clients' ability to interoperate with the developed technology. All of them need to install these pieces of software on clients in order to provide advanced features (and, if Microsoft realized a simplified installation procedure embedded in Windows, its characteristics must be timely disclosed to other server OS producers). In this setting, server-to-server interoperability may be socially beneficial and may make business sense to pursue it. Moreover, reverse engineering may be a legitimate tool for competitors wanting to achieve it, despite the strategies of other competitors. However, there is no legal basis to mandate interoperability. Moreover, in my opinion, also deciding that trade secret may be "forbidden" when interoperability is at stake may be an economically risky undertaking.

Certainly, the Commission's Decision in the Microsoft case made sense, because Microsoft did use tying in order to impose its approach on the workgroup server technology and this fact justifies, *ex post*, also a *disclosure obligation*, which may be *understood as a remedy*. Indeed,

> The Commission also maintains that when a non-Microsoft work group server operating system is installed on a Windows work group server network, it must be capable not only of delivering all its functionalities to Windows client PCs but also of using all the functionalities offered by those client PCs.[187]

And this is precisely what I recommend in this paper – since Microsoft is super-dominant in the market for client PC operating systems. Hence, given the fact that Microsoft embedded in Windows client several functions which may be fully used only by competitors able to interoperate both at client-server and server-server level, the disclosure obligation imposed by the Commission is appropriate. Moreover, in its Decision the Commission also argued that Microsoft had already achieved a dominant (and possibly super-dominant?) position in the workgroup server operating systems market, so that – again – a disclosure obligation may be appropriate (even though – as I have already stressed – I recommend limiting such an obligation to clearly super-dominant players).

However, to conclude this issue, if the Commission decides that firms, which are dominant in a certain market, must always allow interoperability[188] in any adjacent market in which they choose to compete, such a decision – for the sake of clarity – should be stated in precise competition policy guidelines (and possibly in an apposite Notice[189] or even in a Directive). Any other approach is likely to create legal uncertainty and huge administrative legal costs. In particular, any excessively "interventionist" approach is likely to hinder innovation also because competitors – instead of designing their own approach to new growing technologies – will know that, once the market will become profitable enough, the client PC operating systems incumbent is likely to enter and it will be forced to disclose the technical specifications for its own solution.[190]

4.7. Mandating disclosure

In the server operating systems related part of the European Microsoft case, the European Commission clearly showed how crucial access to interoperability information may be for the existence of competition and, in the long run, for the existence of competitive pressure to innovate on platform controllers. The importance of interoperability has been discussed throughout the entire dissertation at hand; hence, I will surely not dispute the Commission's general argument. However, in this paper, I submit that mandatory

[187] *Microsoft CFI*, § 232.

[188] And/or refrain from strategies with the main purpose or effect of preventing interoperability.

[189] As the Commission did in the case of telecommunications.

[190] Again, the Commission may recommend, as a matter of industrial policy, to mandate the disclosure of interoperability information in software markets. I think that markets would indeed benefit from an Interoperability Directive with this aim, despite the fact that the desirability of such an approach is based on an "educated guess" (that I share) more than on undisputable theoretical or empirical evidence. In that context it could be stated that using trade secret in order to prevent interoperability (at least: vertical interoperability) is always a misuse of dominant position, as a matter of law when certain conditions are met.

disclosure/licensing of interoperability information should be limited to the specifications needed to interoperate with super-dominant platforms. The reason I argue in this sense is that, overall, there are several dynamic reasons to argue in favor of competition among standards, instead of favoring common access to an existing standard, as far as some competition seems to be possible at all. This is a point of view that I share with Weiser's competitive platforms model.[191,192] As Weiser puts it,

> even if the industry structure will ultimately rely on a single standard, competition policy should still err on the proprietary side of the line, allowing rival standards to battle it out in the marketplace. To be sure, promoting standards competitions risks forestalling, duplicating, and stranding investment, but even where a single standard ultimately emerges, the temporary competition is likely to produce a better outcome. Moreover, in addition to maintaining the possibility of competition on quality, rival standards also hedge against the risk that the single standard proves flawed in some fundamental matter. Finally, even though some platform standards will appear to be susceptible to the tipping phenomenon, policymakers and courts will not necessarily be able to predict which markets will tip and which standard will emerge as dominant. Consequently, they should encourage rival platform standards, and only where a single one emerges as dominant, facilitate[193] and—if necessary—mandate access to that standard.[194]

Hence, using the wording of Drexel,[195] "competition by substitution" should be encouraged, as far as it has some chance of success. And that means that competition policy authorities should not create the expectation that "competition by imitation" will be artificially eased, as long as a *de facto* standard has not been clearly established. Indeed, the moment in which "competition by substitution" becomes really impossible is not, using the working of European competition policy, the moment in which a dominant position is reached. Indeed, such a position does not exclude that some competition is still possible in the market and, given the dynamic environment in which software competition takes place, some competition today may turn out to be significant competition tomorrow, precisely creating that "hedging" against the risk of being locked in a single standard that Weiser mentioned above. It is, instead, when a super-dominant position is reached that – in the absence of some intervention to allow "competition (also) by imitation", the market risks being simply and fully monopolized. Quoting again Weiser:

> In the event that an information platform owner dominates a market because alternative platforms cannot reach or maintain the necessary critical mass, it will reap a great windfall. For Schumpeterians, this result is acceptable on the ground that the great bounty provided by controlling a proprietary standard may be a necessary incentive to develop killer applications that can support a new platform. On balance, however, it seems more likely that such unmitigated incentives would overreward platform inventors by allowing them to thwart innovation (or reap exorbitant monopoly rents).[196]

Indeed, even though forcing the possibility of competition "within the standard" may, from an *ex ante* point of view, reduce incentives to innovate, I consider that being able to know the specifications in advance (with

---

[191] *See* FARRELL & WEISER, *Modularity, Vertical Integration, and Open Access*.

[192] Here, vertical and horizontal interoperability need not to be treated in the same way. In the first two papers of this dissertation, I criticized Weiser's approach to the issue of vertical vs. horizontal interoperability. The author suggested to allow vertical interoperability as a general rule, but to permit the achievement of horizontal interoperability only in limited cases and once a *de facto* standard had been imposed by a dominant player. Indeed, I am convinced of my critique, because Weiser's proposal had been structured (also) as a "framework to govern intellectual property and Internet policy". Actually, I submit that *self-help*, in the form of decompilation, should always be allowed and the second paper of this dissertation showed that this would not cause major market failures. However, it is in the context of this third paper that Weiser's reasoning – applied as a recommendation for competition policy, instead of intellectual property – becomes appropriate and, in my opinion, should be applied. In other words, Weiser's proposal, understood as a framework for intellectual property, would have limited the possibility of achieving interoperability through reverse engineering, unless super-dominance (*rectius*: a *de facto* standard) had been achieved. In the first and second papers, I already discussed why, in my opinion, these limitations to the freedom of undertaking to perform reverse engineering are not necessary. Instead, Weiser's proposal provides compelling arguments to discuss whether *competition-policy-provided-help* – in the form of an obligation to deal/disclose – should be provided.

[193] Here, the idea of "facilitating" access to a standard, in Weiser's original proposal, encompassed the fact of allowing a certain freedom to decompile, which should have been absent during the war among standards. This is the part of Weiser's proposal that I criticized in the first paper of this dissertation. Indeed, my disagreement is mainly based on an empirical observation: I do not think, and I discussed why, that the freedom to reverse engineer may make reaching a unique platform standard so simple that there is a risk of "entrenching a single standard and precluding valuable competition." (Id., p. 590; see also section "V. Toward a Competitive Platforms Conception of the Legality of Reverse Engineering".)

[194] Id., p. 585.

[195] DREXL, *IMS Health and Trinko*.

[196] FARRELL & WEISER, *Modularity, Vertical Integration, and Open Access*, p. 591.

respect to competitors) and being able to fine-tune them on the technological nature of the dominant player's system are a more than sufficient reward for the design effort of the incumbent.[197] Once again, this is coherent with Weiser's model:

> Stated in competition policy terms, the critical point is that facilitating access to rival platforms, even if anticipated before a firm invests in innovation, will only dull—and not eradicate—the incentives to establish an innovative platform standard. Put differently, economic theory suggests that the rewards of establishing a de facto standard are well above those necessary to motivate parties to champion and develop the standard in the first place.[198]

Because of the aforementioned reasons (i.e. the possibility of indirectly obtaining advantage by the fact of setting a standard, even if third parties are granted access), I would strongly recommend mandating free access to all the interoperability specifications, which are necessary in order to access platforms controlled by super-dominant incumbents, at least when these specifications are protected only as trade secrets (the other potentially relevant case being the one in which patent protection is in place, a possibility that cannot be ruled out, despite the severe contraindications that this kind of legal protection of software may encompass). Indeed, only interoperability specifications need to be disclosed, and not implementations, nor the source code of the proprietary platform. Hence – as I showed in the first papers – this just creates a limited amount of free riding, which a super-dominant incumbent is more than likely to face without major problems. Moreover, this approach would eliminate the need to set a price for disclosure, which is indeed a problematic exercise.[199]

4.7.1. RAND fees

That having been said, I admit that here my point of view on free disclosure is just based on an "educated guess", concerning the necessary level of incentives to innovate; neither does the literature provide much more robust evidence on this. Hence, some prudence may be appropriate and, in principle, mandating a completely free disclosure could be excessively penalizing also for a super-dominant undertaking. In fact, one should consider that if it decides to mandate disclosure, it should also take into account that the current equilibrium of software markets is constructed assuming that reverse engineering is costly. If information is released for free – even if this information was not innovative in itself[200] – several competitive strategies of platform owners may be hindered. Some of these strategies may be socially beneficial, not only increasing profits, but also increasing the value of the software system as a whole for consumers.[201] In these cases, it may be sensible to allow platform owners to charge for the disclosure of trade secrets – even if there are no intellectual property rights protecting them. At least, this may be the case when they demonstrate that they invest(ed) resources in facilitating the production of software which is compatible with the platform (and which will be compatible with competitive platforms as well, in case of disclosure).

In other words, I stress that it is my opinion that free disclosure would not create market failures and I argue that competition authorities, at least when super-dominance is at place, should be allowed to try a similar "educated guess" (indeed, the Commission's "incentive balancing test" requires much riskier bets). However, I concede that it may also be appropriate, as a prudent alternative, to propose the enforcement of a reasonable and non discriminatory (RAND) licensing policy. Knowing, for sure, that

> [u]nlike the concept of optimal price, the notion of reasonable price suggests a range of acceptable values, not a single figure. It is used, especially in the United States, for price setting in patent infringement cases. The principle consists in imagining a hypothetical royalty bargaining between the parties if licensing had been pursued instead of infringement. This hypothetical bargaining is supposed to take place at the date the infringement commenced. The range of acceptable value is then given by the minimum the licensor would have been willing to accept and the maximum the licensee would have been willing to pay. Lawyers

---

[197] This seems to have also been noticed *en passant* by the European Commission in its Microsoft IV Decision (§ 779): "Microsoft has been enjoying a dominant (quasi-monopoly) position on the client PC operating system market for many years. This position of market strength enables Microsoft to determine to a large extent and independently of its competitors the set of coherent communications rules that will govern the de facto standard for interoperability in work group networks."

[198] FARRELL & WEISER, *Modularity, Vertical Integration, and Open Access*, p. 592.

[199] *See*, among many and on this specific case, LÉVÊQUE, *Innovation, Leveraging and Essential Facilities*, pp. 86—90.

[200] For instance, in the context of Microsoft case, Commissioner Neelie Kroes stated (official press release IP/07/269 of 01/03/2007) that: "The Commission's current view is that there is no significant innovation in these protocols."

[201] I discussed some relevant cases in the first paper, § *9.3. A possible economic criticism (IP as a tool enabling desirable business models).*

> conventionally use a list of factors (the so-called fifteen Georgia-Pacific factors) to determine the end-points and a likely outcome of bargaining. The application of this approach [to cases like the Microsoft one] is questionable. Fundamentally, it looks backward whereas the setting of the royalty rate of a compulsory licence has to focus on future. Practically, it requires historical and comparable situations that are difficult to find in the case of Microsoft. Before refusing to supply Sun Microsystems, Microsoft had disclosed information on interface without asking for royalty. Previous agreements cannot serve therefore as precedent. The recent agreement between Sun Microsystems and Microsoft cannot be used either. It terminates a series of different disputes, including Microsoft's infringement of Sun's intellectual property rights. It combines several commitments. The financial transfers are difficult to disentangle within the package.[202]

Moreover, according to the Commission,[203] a "reasonable remuneration" to Microsoft cannot allow the software house to ask a fee "reflecting the strategic value" of avoiding potential competition and keeping its Windows client monopoly or gaining a similar monopoly in the workgroup server market. Hence, if this exercise requires us to imagine that Microsoft had no market power both in the PC operating systems market and in the server operating system markets, it is easy to guess that a "reasonable price" would again be zero or close to zero, because such a firm would evangelize in complementary markets.[204] Indeed, the Commission forced Microsoft to license its interoperability information for an almost symbolic fee of 10,000 €.[205] All that having been said, I would not be as pessimistic as Leveque, when he argues that – in order to determine a reasonable fee – "[a]ccounting methods present the same drawbacks as the past hypothetical bargaining." It is indeed true that "[t]he specific fixed cost incurred by Microsoft in developing interface programs is not kept in book" and that "R&D costs are at best disaggregated per line of products, not per patent and trade secret."[206] However, if the Commission adopted clearer principles and the Microsoft case provided a relevant precedent, knowing in advance that a RAND policy may be imposed, super-dominant players will also be highly encouraged to document their development costs concerning interoperability specifications.[207] Again, making the rules clearer could increase administrability.

Finally, even if one decided that the general rule is RAND policy, this does not mean that in some antitrust cases – like the European Microsoft case – competition authority cannot mandate free disclosure. In fact, the gratuity of disclosure could be an additional sanction for past anticompetitive behaviors and an urgency measure to re-establish a level plain field with competitors (instead of a general duty of dominant agents).[208]

4.7.2. Additional problems with open source licenses

In any case, competition agencies will have to verify that no business model is prevented by the licensing scheme adopted by the dominant player, with particular care for the compatibility of the licensing scheme with an open source model of software development (for the reasons discussed in the previous paper, § 7). Some authors[209] are especially worried that the "EU Commission has indicated that if interoperability information is innovative and covered by patents then compulsory license may not be possible", corroborating this preoccupation with the following quotation:[210]

---

[202] LÉVÊQUE, *Innovation, Leveraging and Essential Facilities*, pp. 88.

[203] *Commission's Microsoft Decision*, § 1008.

[204] However, an ordinary firm could admittedly price at zero just hoping to have a chance to recoup its cost exploiting a future monopoly, exactly what the Commission decision is preventing Microsoft from doing.

[205] See the Commission's press release IP/07/1567, October 22, 2007.

[206] LÉVÊQUE, *Innovation, Leveraging and Essential Facilities*, pp. 88.

[207] There is an additional prudential element that could be considered. Platform controllers may argue that certain interfaces (APIs or CPs) are not disclosed because they are still experimental and the platform controller does not want to generate the expectation that this interface will also be supported in the future. In order to avoid paradoxical results, it may be possible to consider the aforementioned possibility in setting the minimum disclosure requirements between super-dominant portions of the platform and other applications (in particular middleware, but not only). A solution may be that of applying a kind of "most favored nation" clause: if an interface is used by some of the platform controller's separately distributed/sold software, then it must be disclosed. Similarly, an interface must be disclosed if it is used by a portion of the platform competing with pieces of software distributed by third parties as stand-alone programs. In this way, it would be legitimate not to disclose interfaces which are completely "internal" and used by various "obscure" parts of the platform.

[208] See also § *5.2. For true remedies "not immediately" is already "too* late".

[209] See, in particular, MIKKO VÄLIMÄKI, *Software Interoperability and Intellectual Property Policy in Europe*, 3 European Review of Political Technologies, 1—11 (2005).

[210] Commission's press release IP/05/673, June, 6 2005 (available at http://europa.eu/rapid/pressReleasesAction.do?reference=IP/05/673; last visited June 20, 2008).

> Microsoft considers that the software source code developed by recipients of the interoperability information that implements the Microsoft protocols should not be published under a so-called 'open source licence'. The Commission nevertheless considers that, if the Court of First Instance rules in favour of the Commission in the pending application for annulment filed by Microsoft (case T-201/04), *this should be possible for the protocols that do not embody innovations*" (emphasis added).

In my reading, the Commission here was simply arguing that – if Microsoft is able to enjoy patent protection over interoperability information – then it is consistent with EU legislation to ask a reasonable royalty to use it. However, this may be problematic, because – if Microsoft can ask for a per-copy royalty or some similar payment – this would imply the exclusion from the licensing of open source projects, since these projects are based on free distribution and hence incompatible with royalties. In fact, this is the only reason for which it is difficult to imagine how patent protected interoperability mechanisms could be used in open source software. The problem is that this software is typically free, not the fact that its source code is completely disclosed. In fact, patent holders should not care much about the fact that code is open, since they do not need to rely on secrecy, precisely because of the existence of an exclusive right valid *erga omnes*.

For the moment, the Commission has been able to push Microsoft toward accepting (non-disclosure) agreements that required just a one-time compensation to access secret interoperability information. Moreover, the licensing of specific patents does not seem to be strictly necessary in order to implement the specifications concerned by the Commission's Decision.[211] However, in case such a licensing was necessary, allowing also open source projects to deal with Microsoft, under terms that would not prevent the adoption of their model of development, would be important.

4.7.3. A note concerning software patents

If the licensing of software patents is needed in order to achieve interoperability, this may be one of the cases in which a RAND licensing policy may be more appropriate. However, I think that another teaching of Microsoft's cases – concerning software patents the licensing of which is possibly needed in order to achieve interoperability – is worth mentioning. To reduce fear, uncertainty and doubts (FUD), when legal action is threatened on the basis of a patent, the patent number should be clearly stated (as well as the name of the infringing product). In my opinion, competition authorities should consider that – at least for dominant firms – doing otherwise may be considered an abuse of dominant position.

Consider, for instance, the following quotation from Fortune Magazine:

> Microsoft General Counsel Brad Smith and licensing chief Horacio Gutierrez sat down with Fortune recently to map out their strategy for getting FOSS users to pay royalties. Revealing the precise figure for the first time, they state that FOSS infringes on no fewer than 235 Microsoft patents.
> […] Gutierrez refuses to identify specific patents or explain how they're being infringed, lest FOSS advocates start filing challenges to them.
> But he does break down the total number allegedly violated - 235 - into categories. He says that the Linux kernel - the deepest layer of the free operating system, which interacts most directly with the computer hardware - violates 42 Microsoft patents. The Linux graphical user interfaces - essentially, the way design elements like menus and toolbars are set up - run afoul of another 65, he claims. The Open Office suite of programs, which is analogous to Microsoft Office, infringes 45 more. E-mail programs infringe 15, while other assorted FOSS programs allegedly transgress 68.[212]

This is quite interesting, since all aforementioned pieces of software are developed and frequently distributed by several different companies, foundations and individuals. So, not even the identity of infringers could be easily guessed, apart from some. Indeed, some clarity about the patents potentially infringed by a reimplementation of Microsoft's workgroup server related specifications have been identified by Microsoft only after the Commission's *Decision*:

> We have been able once for all to receive a list of the patents that Microsoft claims to be reading on the specifications. Incredibly we have never been exactly told which those patents were. This should be helpful to stop FUD against Samba, and we hope the same will happen with other Free Software projects.

---

[211] In any case, from October 22, 2007, Microsoft provided a particularly low fee optional worldwide patent licence, for a royalty of 0.4 % of licensees' product revenues (see the Commission's Press Releases IP/07/1567 and IP/08/318).

[212] Roger Parloff, *Microsoft takes on the free world*, FORTUNE Magazine, May 14, 2007, http://money.cnn.com/magazines/fortune/fortune_archive/2007/05/28/100033867/ (last visited July 21, 2008).

> It is standard practice: if you have an issue with somebody, you should tell what this issue is, or shut up completely.[213]

Moreover, it can be observed that dominant firms could easily encourage other undertakings to do the "dirty work" of threatening patent litigation, instead of doing that directly.[214] This is why – given fact that listing the numbers of potentially infringed patents does not cost anything to patent holders (indeed, patent protection is not based on secrecy: quite to the opposite, it is conditional on disclosure!) – this kind of rule should be extended to any firm, with particular attention to software platforms markets, because this kind of "announce effect" is particularly strong in complex, modular and incremental innovation based technological environments.[215]

5. An o-ring theory[216] of exclusionary platform behavior

As I discussed, with several references to Weiser's analysis of software platforms, I believe that super-dominant undertakings, having reached and firmly holding a quasi-perfect monopoly in a given relevant market (for antitrust purposes), should be forced by competition policy agencies to disclose the interoperability specifications of their platforms (including operating systems, middleware and any piece of software exposing a significant set of APIs or CPs). For various theoretical reasons, that are summarized in the ICE paradigm of Farrel and Weiser and confirmed by recent theoretical economic developments (as the two-sided markets theory), such a disclosure is already likely to take place. However, competition authorities should be especially ready to impose it – at least – in cases that represent an exception to the ICE paradigm. (Firms thinking that non-disclosure is the line of action a "benevolent monopolist" should take in the ICE setting, may be left free to do so, but bearing the burden of showing why this is the case.) Moreover, I would recommend to extend this disclosure obligation to the file formats of super-dominant applications in general, in order not to allow software houses to lock-in users pivoting on the control of user created content.[217]

The most complex problems may arise when the platform controller also develops a separate piece of software (or, at least, a piece of software that, up to that point in time, was considered to be an independent program). Of course, there may be good reasons for which a platform controller may decide to compete also in complementary markets. However, as the previous part of this paper showed, there are also significant threats to competition on the merits, which may arise in this situation. Indeed, I discussed that there are strong synergies between information-withholding and tying strategies used as exclusionary tools. In fact, using both these strategies, a super-dominant undertaking is likely able to both quickly impose a *de facto* standard to the market (using tying) and keep exclusive proprietary control over it (refusing to deal when interoperability information is at stake).

From the point of view of a competition agency, however, the is also a nice feature of this complementarity (among strategies which just have a modest excluding power if used in isolation). In fact, forbidding just one of them may eliminate or significantly reduce the anti-competitive effects of the dominant undertaking's behavior, typically without preventing the same undertaking from obtaining some legitimate and possibly pro-

---

[213] Carlo Piana, Free Software Foundation Europe, press mailing list, *EU antitrust case over: Samba receives interoperability information*, December 20, 2007 http://mailman.fsfeurope.org/pipermail/press-release/2007q4/000191.html (last visited July 21, 2008)

[214] According to some commentators, for instance, Microsoft would have had an interest in favouring the financial surviving of SCO, which started a famous litigation against some Linux distributors and a worldwide campaign claiming that Linux infringed its own intellectual property. Whether or not these suspicions have some basis, something similar would be possible, in principle, in other cases. *See* Stephen Shankland, *Fact and fiction in the Microsoft-SCO relationship*, CNET News, November 15, 2004 (http://news.cnet.com/2100-7344_3-5450515.html.

[215] Indeed, when this kind of vaporware-like strategy is used by non-dominant firms, one should also consider the possibility of adopting appropriate intellectual property based countermeasures, like a doctrine of patent/copyright misuse, for instance barring legal actions for a certain period (or, at least, assuming the good faith of violators), unless patent numbers of potentially infringed patents has been clearly disclosed by rightholders.

[216] An o-ring is plastic (o-shaped) loop, used as mechanical seal. It is a quite banal item, but a defective o-ring seems to have determined the Space Shuttle Challenger disaster of 1986. (See http://en.wikipedia.org/wiki/O-ring and http://en.wikipedia.org/wiki/Space_Shuttle_Challenger_disaster.) As in the o-ring approach to exclusionary behavior I will discuss, even a small missing or defective part could prevent the functioning of a complex system. For my purposes, the two key elements of an exclusionary strategy concerning complementary products are tying and information-withholding: preventing just one, the entire exclusionary strategy will be disrupted.

[217] See § 3.3: market power associated to the control over file formats is frequently associated to a huge amount of user's performed investments in these files (with associated lock-in) and it may have nothing to do with the technical quality of the file format itself.

competitive goals. More specifically, my recommendation to competition authorities is to take modularity as a guiding principle, simply because – given the high degree of complexity of today's software systems – engineering and design related considerations suggest that favoring it will not distort technological development too much. (In other words, I suggest that software is already developed in a modular way for technical reasons and that the absence of modularity is typically artificially created. In effect, the proposal that follows could be turned upside down, taking disclosure as a guiding principle and modularity as a subsidiary one, but I believe that favoring modularity is less distortive of existing business models in the software industry.) My proposal may be summarized as follows.

Obviously, non-dominant undertakings should be free to design their products as they like. Moreover, I suggest that also undertakings with a dominant position (but not yet "verging on absolute monopoly") should not be limited in their design choices, because of the benefits of competition among standards I summarized referring to Weiser's work. However, dominant firms should be aware of the fact that – if their products reach a super-dominant position – some constraints will arise (and they will not be excused if they deliberately decide to ignore this predictable situation on their "path for super-dominance"[218]). It is only when an undertaking becomes super-dominant that a duty arises to apply the principle of modularity, which, however, will not be an absolute duty. In fact, as long as the principle of modularity is respected, super-dominant undertakings will be free to use secrecy to protect their own complementary innovations, despite the fact that secrecy may sometimes be inefficient from a social point of view. (Indeed, intellectual property itself is frequently inefficient, but any market-based tool to finance innovation creates static inefficiency while "extracting surplus".) However, if an undertaking decides to tie new functions to its super-dominant product, it will be free to do so only as long as the alternative principle of disclosure (or full interoperability, if you like) is respected.

Moreover, notice that modularity does not prevent the free distribution of complementary products, nor the availability of packages including both the platform and the complementary products. It just imposes the existence of unbundled packages, in which the super-dominant platform is available as a standalone piece of software (and at a cost which is not higher than the bundled package: I will discuss further this condition later). Indeed, this result may simply be achieved (as it is done by Linux distributions and, in part, by any platform producer) by allowing users to check, from a list, which additional software they want to install, in addition to the "mandatory" core of the platform.[219] Moreover, it must be possible, both for final users and OEMs, to remove the modules from the platform without any legal or strategically imposed technical obstacles.

For instance, assume that the European Commission had clearly taken modularity as a guiding principle in a specific guideline to summarize the special responsibility of super-dominant undertakings in software markets. In that case, as the Commission imposed in its Decision, Microsoft would have been forced to release a version of Windows without WMP (and without Internet Explorer, Outlook Express, Messenger and similar complementary software) and/or to allow users to uninstall these pieces of software. Clearly, the first time this modular approach is imposed, it may require some significant costs to modify Windows, but – when the principle becomes clear – there are essentially no technical reasons for which Microsoft should not be able to design Windows in a modular way. (Indeed, during the US Microsoft case expert consultants showed that Windows is already a modular system, able to perfectly work without supposedly "essential components" as Internet Explorer.[220]) Assume, however, that Microsoft argued that the tying of WMP with

[218] In other words, dominant undertakings should take into account that they will face constraints if and when they become super-dominant. It should be clear that, once an undertaking becomes super-dominant, saying that a given design choice had been take when super-dominance was not yet achieved and that this created lock-in in this choice will not be an acceptable excuse.

[219] In order for this approach to be equivalent to the distribution of the platform as a standalone product, it must also be possible to automate the choice in case several installations are performed (because this is useful to OEMs), but this is technically banal (e.g. a text file with the list of wanted/unwanted complementary products may be saved/checked by the installation program). Or, even better, a small program to "repackage" only the wanted parts of the system may be offered to OEMs (indeed, third parties already realized such a simple piece of software for users wanting to "tweak" Microsoft Windows: see, for instance, www.nLiteOS.com).

In the European Microsoft case the requirement of physically packaging a version of Windows without Windows Media Player can be described either as a symbolic/emblematic remedy or, if one does not especially like the European Commission, as a marketing choice of the Commission to make its decision much more spectacular are easy to turn into effective newspaper headlines. Indeed, only a few copies of this version of Windows have really been sold to final customers.

[220] Obviously, some portions of Windows, like the users' guide, may not work if a browser is not installed, and possibly also if Internet Explorer specifically is not installed. This problem – and the problem of "dependencies in general" – could be solved if Microsoft itself proceeded to allow the uninstall of IE. In any case, further on in the reading of my proposal, one will notice that there is also another potential solution to this problem.

Windows is a key element of its strategy to increase the availability of media players and ensure developers that Windows will carry fully working multimedia capabilities. In this case, the competition agency may allow Microsoft to bundle WMP with Windows, as long as all the APIs used by WMP are disclosed to Microsoft competitors, all the specifications of WMP streaming technology are disclosed and the same happens for embedded DRMs. Indeed, in the case of WMP, Microsoft will likely opt for modularity, instead of disclosing all the specifications of these proprietary technologies, however disclosure may be a preferable alternative for Internet Explorer (IE) or other "complementary products", which may be more genuinely described as "components" of Windows. In the case of IE, tying it to Windows would force Microsoft to document all non-standard technologies embedded in Internet Explorer, so that competitors could decide if they wanted their browser to be compliant with ordinary World Wide Web standards and/or with Microsoft standards.

Of course, the complementary products, which become tied to Windows, will also have a significant chance of becoming widespread and potentially a *de facto* standard. However, the disclosure obligation, constituting the condition for the legality of the violation of the modularity principle, will ensure that "competition within the standard" will be an available opportunity for competitors. In this way, the leveraging of monopoly power to adjacent markets will be significantly hindered. Moreover, some minimum principles of modularity or, I should better say, users' choice should be guaranteed: components tied to Windows should not be "imposed" on users. For instance, it must be possible to eliminate icons from the desktop, start menus and similar places; additionally, competition authorities may mandate the possibility of uninstalling at least the user interface of these pieces of software, giving the possibility of setting alternative products as "*default*". In the specific example of Internet Explorer, the browser engine could still be used by Windows's user guide, by Microsoft's or third parties' file managers and so on, but the user should not be unnecessarily haunted by sessions of the browser popping up without any technical need.

The same approach could have been followed by the Commission in the workgroup server related part of the decision. The alternative offered to Microsoft by the European Commission could have been: do you want to tie new network-related functionalities to Windows, where you have quasi-monopoly? Then you have to disclose the communication protocols between these new pieces of Windows and your server operating systems. If you do not want to disclose your CPs (for instance because you think that this would allow competitors to free-ride on your costly software design investment), you may just leave these functionalities free to download and you are not forced to disclose the protocols between the Windows add-ons and the server. All you have to disclose – according to the general duties of a super-dominant undertaking – are the APIs connecting Windows and the new functionalities, which are considered almost as if they were part of the "server" system, where you are not super-dominant.

In other words, disclosure obligations are limited to markets were an undertaking is super-dominant and no competition is possible, unless it takes place "within the standard", however when one ties something to these markets, then the disclosure obligation is extended to this "something". Hence, the dominant platform controller may choose between the two strategies (possibly adopting a combination of them for each one of the newly introduced functions): unbundling from the software platforms of the newly introduced parts that incorporate undisclosed interfaces (APIs or CPs); or the mandatory disclosure of these protocols. What the super-dominant agent will actually decide to do depends on various factors. In particular, on the one hand there may be arguments against unbundling: for instance, when the new function has not been artificially bundled with the platform, but for some technical reasons it is an integral part of it. In this case, unbundling would be technically inefficient and would not serve any consumer interest. However, the fact that it is not economically viable or technically efficient to separate the code devoted to the newly implemented function from the platform is also an argument supporting the view that the overall system (now including this function) is actually an essential facility for everybody wanting to build on or communicate with the dominant platform, so that the disclosure duty looks advisable. On the other hand, there may be arguments against disclosure: if all that is really needed to interoperate with the core functions of the dominant platform is already available, imposing obligation disclosure on independent pieces of software just because they are realized by the dominant platform's owner would simply allow competitors to free ride on the leader's investments.

In particular, the Commission's disclosure obligation in the European Microsoft case may be criticized arguing that competitors were actually able to realize new programs to install in Windows, to replicate what Microsoft did, in order to make available to users the client-side of its workgroup technology. Hence, one may conclude, there was no reason to disclose anything more that what Microsoft had already disclosed. Anything further would just have helped Microsoft's competitors, giving them the possibility of free riding on

part of the leader's investments. In other words, according to Microsoft this was the case for the workgroup functions embedded in Windows (clients): allowing competitors to use the client part of Microsoft's workgroup system would have been equivalent to allowing competitors to free ride on the client part of the system (while receiving also some hints concerning the server part). Indeed, I do not know if this was really the case, but my proposal is robust to this kind un uncertainty. Should this objection make economic/technical sense, then the unbundling remedy will frequently be chosen, since firms may not want competitors to free ride on their investments. In any case, undertakings may be quite confident that consumers will prefer a complete version to an incomplete one, so releasing two versions of a platform (bundled and unbundled) and/or allowing users and OEMs to uninstall components they don't like should not change their choices.

Finally, it is also perfectly sensible, for the platform controller, to modify its strategy over time (for instance offering innovative technologies as stand alone software, with undisclosed characteristics, but then bundling them and disclosing their APIs or CPs, when these technologies become widely used and demanded by almost all customers) or to realize a bundled basic version (with fully disclosed APIs and CPs) and an unbundled advanced version (not subject to the obligation of disclosure). Moreover, in several cases, platform producers may choose a mid-way solution, integrating into the operating systems those parts of an application that expose some relevant APIs and effectively work as part of the operating system and not the users interface or other specific parts. Again, this is consistent with the kind of unbundling obligation that the European Commission imposed in the Microsoft case:

> [T]he impugned conduct concerns only the application software that constitutes Windows Media Player, to the exclusion of any other multimedia technology in the Windows client PC operating system, and the basic multimedia infrastructure of that system remains in the version of Windows imposed by Article 6(a) of the contested decision. It was also [already] stated […] that Microsoft itself differentiates in its technical literature the files which constitute Windows Media Player from the other multimedia files, notably those relating to the basic multimedia infrastructure.[221]

5.1. The cost of errors

The approach proposed in the paper at hand is relatively robust in terms of error costs. It is well known that antitrust decisions may create welfare losses both in case of false positives and false negatives.[222] In particular, I follow White[223] in considering that the different risks and effects of a false positive and a false negative should both be carefully considered and weighted, without assuming that the first or the latter are necessarily the ones to avoid for some abstract (I would say: ideological) reasons.

First of all, in the case of the technological tying of new functions with dominant software platforms, one should consider that modularity is quite a common characteristic of modern software development. Indeed, that Windows itself is – from a technical point of view – already almost completely modular is also confirmed by the fact that

> throughout the period between June 1998 and May 1999, when Microsoft first integrated WMP 6 in its Windows client PC operating system without allowing OEMs or users to remove it from that system, Microsoft offered its streaming media player as separate application software, without any effect on the functioning of the Windows operating system.[224]

Also notice that there are small third party applications (an example is NLite[225]), which could be used not to install some of the "essential" components of Windows, which users (and OEMs!) are normally unable to eliminate from their computers: Media Player, Internet Explorer, Outlook Express and other programs which cannot be removed from the system or are only apparently "removed" (in reality, they are just hidden to the user, but remain installed in the system).

---

[221] Microsoft CFI, § 1164.

[222] "False positives" concern the finding of anticompetitive conduct when, in fact, there is a welfare enhancement; "false negatives" concern an erroneous finding that a conduct is not anticompetitive. See, in particular, DAVID MCGOWAN, *Between Logic and Experience: Error Costs and United States v. Microsoft Corp*, 20 Berkeley Technology Law Journal, 1185 (2005); BARBARA ANN WHITE, *Choosing Among Antitrust Liability Standards Under Incomplete Information: Assessments of and Aversions to the Risk of Being Wrong*, 20 Berkeley Technology Law Journal, 1173 (2005). See also MELAMED, *Exclusionary Conduct*.

[223] WHITE, *Choosing Among Antitrust Liability Standards*.

[224] *Microsoft CFI*, § 1165.

[225] See http://www.nliteos.com (last visited June 20, 2008).

Hence, the risk of forcing modularity when integration would be superior is likely to be very low; moreover, the cost of such an error is likely to be negligible in terms of technical performances, given the pace at which the performances of hardware progresses. Moreover, if the cost we are talking about derives from the lack of coordination of users' choices and/or lack of network effects, OEMs and users' themselves – through the use of the Internet – will likely be able to rapidly offset any "inefficient unbundling". Moreover, any risk is additionally lowered by the fact that integration may still be chosen by the firm, as long as all interface information is completely disclosed. Ayres and Nalebuff[226] share this intuition, commenting the Commission's Decision about the unbundling of Windows Media Player, arguing that:

> What is remarkable is that the [Commission's] remedy [concerning WMP] eliminates false positives [*rectius*: it has zero cost in case of false positives]. Just as King Solomon's proposal to divide the baby only caused pain to the true mother, the Commission's remedy will only cause pain to a monopolist who abused its position.[227]

Even though I consider such a statement slightly overbroad (because some technical and economic cost of forcing modularity over integration may, indeed, exist), this is a point worth reflectin on. It is indeed true that, as the authors noted,

> [i]n criticizing the proposed remedy, Microsoft fatally undercut its argument for the stay. Microsoft claimed that there would be little commercial demand for the "Article 6" version of Windows without WMP. As the Court of First Instance recognized in its decision, that claim was fundamentally inconsistent with Microsoft's argument that imposing the remedy would lead to a high probability of serious and irreparable harm. An ineffective remedy is unlikely to cause irreparable harm.[228]

Moreover, as I already observed, OEMs will not ship "suboptimal" copies of Windows, i.e. copies without the best available free media player or other complementary software.[229] However, that just means that no costs are likely to arise because of the existence of copies of windows without appropriate middleware. In principle, nothing guarantees that there are no technological synergies in designing Windows and WMP as integrated products. If that was the case, these synergies could – in principle – justify the technological tying (if generated by the tying itself). In the Microsoft case, however, Microsoft did non present any evidence in this sense:

> as the Commission observes both in the contested decision and in its pleadings, Microsoft does not show that the integration of Windows Media Player in Windows creates technical efficiencies or, in other words, that it 'lead[s] to superior technical product performance' […].
> In the reply, Microsoft asserts, for the first time, that 'Windows […] operate[s] faster when media functionality is integrated'. In that regard, it is sufficient to state that that assertion is unsupported.[230]

5.2. For true remedies "not immediately" is already "too late"

The antitrust policies suggested in this paper, if coupled with "high enough" fines against violators, could *deter* super-dominant software houses from abusing their market power. To be "high enough", fines need to have an expected value, which is equal to (or slightly higher than) the expected profits that the incumbent may receive from the forbidden combination of information-withholding and tying strategies. Clearly, to achieve such an outcome, antitrust agencies may work on the traditional levers of the probability and level of the fines. But the inequality may hold even reducing the expected profits from the violation: this may be achieved either reducing the duration of such a violation or improving the communication concerning the likely interventions of the agency. In fact, in software markets and in other markets interested by high network effects, market shares gained are precious and very sticky above certain thresholds;[231] hence, it is

---

[226] AYRES & NALEBUFF, *Going Soft on Microsoft?*.

[227] Id., p. 9.

[228] Id., p. 9.

[229] *See* also *Microsoft CFI*, § 1155: "[C]onsumer demand for an 'out-of-the-box' client PC incorporating a streaming media player can be fully satisfied by OEMs, who are in the business of assembling such PCs and combining, inter alia, a client PC operating system with the applications desired by consumers (recitals 68 and 119 to the contested decision). Nor does the contested decision prevent Microsoft from continuing to offer the bundled version of Windows and Windows Media Player to consumers who prefer that solution."

[230] *Microsoft CFI*, § 1159—1160.

[231] Cabral labels these thresholds "absorption barriers": once the threshold has been reached, the market almost irremediably "crystallizes" on a given technology. See LUÍS CABRAL, *Economia Industriale*, (Carocci, Rome. 2002), pp. 385—390.

useful to intervene as fast as possible in case of violations. Moreover, agents' choices are highly dependent on expectations and so is the effect of any market strategy, including abuses of dominant positions. If antitrust authorities are able to make clear that they will stop any attempt to leverage monopoly power to adjacent markets, strategies with such an aim will be less credible, thus less effective. This is why creating clear rules and unambiguous precedents may have a double effect. Not only the likelihood of deterrence will be increased directly (i.e. convincing dominant undertakings that abuses will be punished), but also indirectly (i.e. reducing the effectiveness of abuses in convincing other market players that abuses will not work as well as in the absence of antitrust intervention).

However, when *deterrence* does not work for some reason, the measures I suggested – no matter how high are the fines for violation (apart from cases in which the violator is quasi-bankrupted) – are far from being sufficient in order to *restore competition*. That should be clear, by now, to antitrust authorities: Netscape has been annihilated by Internet Explorer, RealPlayer marginalized by Windows Media Player, workgroup servers are likely to be dominated by Microsoft (even though, in this market, the intervention of the European Commission probably predated the achievement of super-dominance by Microsoft, so that some chances that competition may slowly increase still exist). As a remedy to restore competition, the obligation to sell copies of Windows without WMP preinstalled ("Windows XP N") has been ridiculous: during the first nine months of availability, 1,787 copies of Windows XP N have been sold (on 35.5 million of copies of Windows XP) and no OEM have ordered or preinstalled this system.[232] This is why it is so important to give to the Commission's Decision the status of important precedent (or, at least, of important guiding principle), otherwise it will be completely empty of any meaning.

In terms of efficiency and social welfare, the fact that Microsoft or other incumbents are able to substitute other market leaders may not have such dramatic consequences. Indeed, there may even be some static efficiencies. However – in dynamic terms – if Microsoft knows that, risking some hundreds of millions of dollars – a market may be effectively monopolized, there are cases in which such a deal may sound quite a bargain. And that is the case if conquering a certain market effectively protects the core of Microsoft's profits, i.e. the monopoly in client operating systems. (But think also about other exceptions to the ICE paradigm.) If what is ultimately at stake for Microsoft is the Windows monopoly, no fine from the European Commission could deter abuses. Indeed, in 2007 Microsoft's total revenues accounted $51,122 millions[233] and the operating income generated by the client segment at $11,603.[234] In other words, Windows-related *profits* reached about 23% of Microsoft *total revenues*. Note also that "the OEM channel accounts for approximately 80% of total Client revenue",[235] so confirming that pre-installed copies of Windows have a lion's share in Microsoft's profits, which frequently mimics the movements of personal computers sales.[236] Similar data applied also at the time of the *Commission's Microsoft Decision*.[237] Compare now these amounts with the "record fine" imposed by the European Commission to Microsoft in 2004, which amounted to "just" 497 million euro (less than 615 million dollars at the time). This fine has been followed by additional ones because of Microsoft's non-compliance (280.5 millions euro in 2006 and 899 millions euro in February 2008). However, consider also that the first complaint from Novell about Microsoft's interoperability-information-withholding

---

[232] See Microsoft's *Fact Sheet: Windows XP N Sales*, April 2006 (available at http://www.microsoft.com/presspass/legal/european/04-24-06windowsxpnsalesfs.mspx; last visited June 15, 2008).

[233] *See* Microsoft's Annual Report, available at http://www.microsoft.com/msft/reports/ar07/staticversion/ (last visited July 20, 2008).

[234] *See* Microsoft's Fourth Quarter FY 2007 Earnings Release (and related documents), available at http://www.microsoft.com/msft/earnings/FY07/earn_rel_q4_07.mspx (and related links) (last visited July 20, 2008). For fiscal year 2007, Microsoft (unaudited) segment operating income (Loss) in millions dollars where the following: Client 11,603; Server and Tools 3,900; Online Services Business (732); Microsoft Business Division (including the Microsoft Office Suite) 10,838; Entertainment and Devices Division (1,892); Corporate-Level Activity (5,193). The total operating income was 18,524.

[235] *See* Microsoft's letter to shareholders attached to Microsoft's Fourth Quarter FY 2007 Earnings Release (*supra* note 234).

[236] *See* Microsoft's letter to shareholders attached to Microsoft's Fourth Quarter FY 2007 Earnings Release (*supra* note 234): "Client revenue growth correlates with the growth of purchases of PCs from OEMs that pre-install versions of Windows operating systems".

[237] See, in particular, footnote 1342 of the Decision: "Microsoft is currently the largest company in the world in terms of market capitalisation. […] Microsoft's resources and profits are also significant. Microsoft's Securities and Exchange Commission filing for the US fiscal year July 2002-June 2003 reveals that it possessed a cash (and short-term investment) reserve of USD 49,048 million on June 30, 2003. As regards profits, this Securities and Exchange Commission filing indicates that in US fiscal year July 2002-June 2003, Microsoft earned profits of USD 13,217 million on revenues of USD 32,187 million (profit margin of 41%). For the Windows PC client PC operating system product during this period (Client product segment), Microsoft earned profits of USD 8,400 million on revenues of USD 10,394 million (profit margin of 81%)."

started in 1993 and was reinforced by Sun Microsystems's claims in 1998. Moreover, in 2004, waiting for the European Commission's decision, Microsoft's liquidity was about 53 billions dollar.[238] In such a scenario, "investing" every few years a few hundreds millions dollar in antitrust liability may indeed be a good business strategy, provided that this makes credible the threat to extinguishing any competition, which may potentially threaten the Windows-related rent of Microsoft. And not even imposing, each year, a fine achieving the theoretical maximum of 10% of Microsoft's profits could one be sure to exercise an appropriate *deterrence*, unless leveraging strategies are stopped fast enough to be made ineffective.

It may indeed be interesting to spend a few more lines about fines in the European Microsoft case. The fine appears to have been calculated in the following way.[239] The Commission took "7.5% of Microsoft's overall EEA turnover on the markets for client PC operating systems and work group server operating systems in the business year ending 30 June 2003" as the initial amount.[240] That totalled to EUR 165,732,101. Then it applied a "weighting of 2 to that amount to ensure that the fine was sufficiently deterrent and to reflect Microsoft's significant economic capacity".[241] Finally, "as regards the duration of the infringement", the Commission increased by "50% […] the basic amount of the fine"[242]. The Court of First Instance found that this fine is not excessive, nor disproportionate to Microsoft's violations.[243] That is fair enough. It is less clear, however, how such an amount could "ensure a sufficient deterrent effect on Microsoft". It is possible to assume that the 7.5% of the turnover is somehow related to the "return" of the antitrust violation; alternatively, it may be an amount "going toward" the maximum 10% threshold, but remaining below, probably because of some principle of proportionality, since – after all – Microsoft did not engage in some of the most despised antitrust violations, like naked collusion. In any case, this 7.5% is a quite an arcane number, that does not seem to have been chosen because of its deterring characteristics. However it is even more puzzling that a violation, which had already lasted at least "5 years and 5 months" (at the time of the Commission's Decision), should be deterred by a fine calculated on the turnover of one year times 1.5, instead of times 5 or more.[244] Apart from that, in general one would appreciate some more explanations of the economic reasoning behind the Commission's estimate of an appropriate (if not optimal) fine

All that considered, I am not surprised that "Microsoft was the first company in fifty years of EU competition policy that the Commission has had to fine for failure to comply with an antitrust decision".[245] Moreover, one has to be aware that both in the media player and in the workgroup server markets Microsoft's exclusionary practices had to stop, but nothing in the Commission Decision could restore the level of competition at a pre-violation stage. Given the fact that this paper was focused on the kind of limits that competition policy should pose to the freedom to design software products and/or keep secret their interoperability characteristics, I may largely approve the Commission's Decision. However, in software markets, after a 60% or maybe 80-90% market share has been reached, we should never ask ourselves if a given approach or non-structural remedy could "restore competition". It simply cannot.[246] No free disclosure imposition, no unbundling remedy, no behavioral rules could make agents ignore actual market shares. In the server market, one may consider that the fact that the Commission extended Microsoft's obligation to disclose to server-to-server interoperability and forced Microsoft to license its trade secret almost for free (a flat rate or 10,000 €, which is purely symbolic) as a kind of additional remedy going in the direction of restoring competition. However, this is far from being sufficient. In the media player markets, as mentioned quoting Ayres and Nalebuff, at least an additional must-carry obligation concerning competitors' products should have been imposed.

At the end of the day, once violations have been performed, "restoring competition" is a problem that can be addressed – if at all – using structural remedies (or behavioral remedies bordering regulation). This is why

---

[238] See Robert Barker, Why Microsoft's Cash Makes It A Bargain, BusinessWeek May 3, 2004 (available at http://www.businessweek.com/magazine/content/04_18/b3881134_mz026.htm; last visited August 2, 2008).

[239] Microsoft CFI, §§ 1360—1364.

[240] Microsoft CFI, § 1360.

[241] Microsoft CFI, § 1363.

[242] *Microsoft CFI*, § 1364.

[243] *Microsoft CFI*, § 1366.

[244] See the *Commission's Microsoft Decision* at §§ 1077—1078.

[245] See the Commission's Press Release IP/08/318, February 27, 2008.

[246] About the inefficacy of remedies in the European Microsoft case, as tools to *really restore competition*, see FIRST, *Strong Spine, Weak Underbelly* and FIRST, *Netscape is Dead.*

it is crucial, for the future, to build fines with optimal deterrence in mind and, above all, to create strong precedents, generating clear competition policy principles, leading to quick and predictable decisions.[247]

6. Zero price is a constraint on anticompetitive behaviors

Describing my recommended policy, I argued that – in order to respect the principle of modularity – it is sufficient to make available to users and OEMs an unbundled version of the platform (and, possibly, of the complementary product, so that consumers choosing the unbundled version know that – if they like – they can "re-create" an integrated version just installing the complementary product). In order for the principle to be respected, it is sufficient that the cost of the unbundled version is not higher than that of the bundle. (Notice that imposing on consumers artificially high research cost for finding the unbundled version would violate this condition: the unbundled platform must be really available.) Even in cases in which there is no demand from distribution, the platform controller should, at least, ensure online availability through e-commerce (in fact, the actual availability of the unbundled version is clearly a special case of this general condition: if obtaining an unbundled version costs effort, then the bundle is practically cheaper). Similarly, the super-dominant platform controller will not be allowed to offer discounts on the price of its own platform conditional on the number of complementary product installed or other policies, which are *de facto* equivalent to a negative price of the complementary good.

The previous condition, however, is prone to at least two critiques. The first one concerns the fact that a zero price for the complementary product would allow the platform to perform a kind of "predatory practice". The second one, probably even more significant, concerns the fact that – if it is possible to practice a zero-pricing policy – all incumbents would always choose modularity instead of disclosure, when adding new functionalities to their platforms, since – after all – consumers will prefer the "complete" version to the unbundled one, if the price is the same.

There are two categories of answers to these critiques. First of all, it can be shown that there are indeed several cases in which a zero price for a software program may be a profitable strategy, so that competitors are not likely excluded by this policy. That having been said, it is also possible to demonstrate that there are cases in which a non-negativity constraint on the price of the complementary piece of software is already quite a binding constraint for a platform controller, so that imposing more stringent constraints may negatively affect social welfare.

As far as the first point is concerned, that software may be priced at marginal cost (which is essentially zero[248]) is verified by experience. This is true, in particular (but not only), for free open source programs and for cohorts of freeware programs distributed on the Internet, both in order to collect advertising revenues or in order to boost the creation of network effects, generating revenues in other markets. (See, for instance, the aforementioned example of Acrobat Reader and the correspondent "writer" program from Adobe, or media players, were revenues may come from advertising, sales of music, sales of the server-side of the technology to content creators and distributors. But think also about other business models, based on the offer of customizations, assistance, learning facilities of various kinds). Indeed, the true marginal cost of software may be negative, since the production and distribution cost is almost zero and each additional copy creates network effects[249] and/or advertising or other revenues. For instance, Microsoft is not charging for WMP; however, the software house is allowing the update to Windows Media Player 11 only for people having a legitimate copy of Windows,[250] hence, the fact that Microsoft dominates this market with the "standard"

---

[247] See also KÜHN & REENEN, *Interoperability and Market Foreclosure.* (, pp. 28—29. "Much of the positive impact of competition policy is through deterring anti-competitive behaviour without the need for ever taking legal action."

[248] Notice that also arguments about the fixed cost of distribution (e.g. need of powerful servers, etc.) are no more strictly compelling. In fact, technologies like the peer-to-peer ones (e.g. BitTorrent) make the price of distributing a huge number of copies even smaller than the cost of distributing a few copies, since downloaders actually contribute to provide bandwidth to other downloaders. Hence, the worldwide distribution of a software may, in principle, be performed investing something like 10 $, in order to register a domain and post a tracker, pointing to some initial copies of the software to distribute for free.

[249] This effect is particularly evident in two-sided markets model: see, for instance, the model of ROCHET & TIROLE, *Two-Sided Markets: An Overview*.

[250] Interestingly, the latest version of Internet Explorer (IE7) may also be downloaded by people that possess a pirate version of Windows. In fact, the market leadership of IE seems to be increasingly disputed by competitors such as Firefox and Opera and browsers are a much more dangerous middleware than media players, at least for the time being. Hence, it is reasonable to suspect that Microsoft did not want to renounce to any possible network effect generated by copies of Internet Explorer, not even by copies installed on pirated copies of Windows. (Any Windows user may verify for which software downloads Microsoft requires the acceptance of the installation and control of Windows Genuine Advantage: for privacy reasons, also legitimate users may

media player may increase the likelihood that users buy a genuine copy of Windows, instead of a pirated one.[251]

Consider also that the idea that the "complete" version of a platform will always be preferred – so that, at most, competitors will be able to enjoy a fraction of the network effect of the platform controller[252] – is not necessarily true either. First of all, there may be reasons for avoiding some of the incumbent's complementary products: for instance, they may be prone to security vulnerabilities and – if one does not use them – it is better not to have them installed on the PC at all. Moreover and more importantly, it must be recalled, as observed (among others) by Ayres and Nalebuff, that in cases like the one of Windows Media Player,

> the real customers are not end users, but rather Dell and other OEMs. Rival players may provide financial incentives to these OEMs to establish their player as the default option. (The rivals can recover those payments through upgrades, or through the sale of subscription services marketed through the player.) End users, too, will benefit, as some of these incentive payments will be passed on in the form of lower hardware costs. Thus OEMS and end users may choose the unbundled version of Windows because, after incentive payments, it is cheaper.
> Note, though, that Microsoft can't play the same game. They can't offer OEMs or end users a financial incentive to make WMP the default, or engage in any similar machinations. They are specifically prohibited by doing so by the Commission's Order as that would make the bundled version less expensive than the version without it. Indeed, this restriction on pricing may be the most biting part of the remedy.[253]

All that having been said, in this section of the paper I am also going to propose a simple model, showing additional reasons because of which a non-negativity constraint on the price of complementary goods may, indeed, by binding.

6.1. Complementary oligopoly model

Depoorter and Parisi (2003) applied a model of complementary oligopoly pricing to the market for intellectual property rights, confirming various intuitions coming from Cournot's models and from the literature on double marginalization: "*In the market for complementary goods, price coordination and monopolistic pricing do not necessarily represent inefficient equilibria, when compared to the alternative Nash equilibrium*". Intuitively, the result can be imagined as a generalization of the more familiar double marginalization problem, arising in case of two monopolies in a vertical chain of distribution/production. In fact, because of externalities generated by independent pricing choices of complementary goods producers, also from the point of view of social welfare

---

choose not to install WGA, in principle, and this precludes them some updates. *See* also http://en.wikipedia.org/wiki/Windows_Genuine_Advantage (last visited August 20, 2008): "Microsoft includes the Windows Genuine Advantage Validation Library in several products like Windows Defender, Microsoft Private Folder and Windows Media Player 11 to validate the Windows installation. Internet Explorer 7 no longer requires the user to pass a Windows Genuine Advantage test in order to download or install the software. Older versions, however, did.")

[251] Indeed, improving the efficacy of this strategy could have even been an additional reason to monopolize the media player market. In fact, it is possible to apply here the intuition of Carlton and Waldman regarding the fact that a monopolist may choose to exclude rivals in order to be able to capture the value of upgrades of the complementary product. *See* DENNIS W. CARLTON & MICHAEL WALDMAN, *Tying, Upgrades, and Switching Costs in Durable-Goods Markets*, NBER working paper series, n. 11407 (June, 2005). If, to have "the standard" media player, one had to buy a legitimate copy of Windows, Microsoft could increase the efficacy of its discrimination strategy among legitimate users and pirates. Indeed, it is plain that Microsoft does not want to strictly enforce its intellectual property rights on Windows against end users' piracy. In fact, Windows does work even if users do not have a legitimate key to activate it (this choice likely being related to the need of maintaining the broadest possible installed base in order to keep intact direct and indirect network effects). Indeed, Windows Genuine Advantage is used by Microsoft more as a tool against distributors deceivably selling pirated copies of Windows to unknowing users than as a tool to hinder the working of pirated copies of Windows. See in particular Microsoft's press release *Microsoft Files Lawsuits to Protect Consumers and Software Resellers*, Sept. 19, 2005 (available at http://en.wikipedia.org/wiki/Windows_Genuine_Advantage#Circumvention; last visited August 20, 2008). However, pirate copies of Windows are slightly disadvantaged in terms of non-security-related updates, a typical example of which is WMP. (Security updates are allowed, because, as for a vaccine against real world viruses, there are reasons to let also pirates avoid malware: this decrease the likelihood of a contagion also for legitimate users.) In practice, Microsoft is price discriminating, offering *de facto* free copies of Windows to (home) users accepting to use pirate software. (For business users there is at least some risk of being detected by traditional controls of law enforcers when using pirate software: in practice, this risk is null for home users, unless Microsoft decided to use technological measures in order to do so.)

[252] This could be assumed if a "complete platform" were always preferred. In this case, the platform controller would be sure that a copy of its own complementary good would be present on each computer, even though some users could "multi-home" (using a two-sided markets wording) adopting more than one complementary product.

[253] AYRES & NALEBUFF, *Going Soft on Microsoft?*, p. 7.

"*price coordination and monopolistic supply often constitute an improvement over the alternative equilibrium outcomes*". As highlighted by the authors, this model has "*significant and obvious implications* […] *for antitrust regulation*" in the field of intellectual property when dealing with complementary goods. Moreover, I will show that it has slightly less obvious implications in terms of antitrust remedies.

This section applies some intuitions coming from the previous models concerning complementary oligopoly to the European Commission case against Microsoft.[254] The analysis will use as an example the tying of Microsoft's Windows Media Player (WMP) with Microsoft Windows operating system (OS). However, as I discussed, a similar reasoning could apply to the tying with Windows of client-side technologies used by Microsoft's workgroup server operating systems. Starting from an insight provided by Hagiu (2004), the aforementioned model is modified, allowing Microsoft to commit to a certain price level of Windows: in this way, Microsoft acts like a leader in a Stackelberg duopoly model, while producers of complements act as followers (despite the fact of being present in their own markets even before Microsoft).

The approach adopted is the following: in an "initial game", Microsoft commits to the price of Windows, and then an independent developer chooses the price of its complementary product. After some time[255], a "second game" is played and Microsoft (as any other potential entrant) becomes able to realize a functional copy of the original complement (and it can choose, in this "second game", to produce it and eventually bundle it with Windows). As the model will show, in the "second game", Microsoft (but no other potential entrant) has an incentive (at certain conditions) to enter in the complementary market and drive down the price of the complement to zero, in this way reaching an equilibrium which is as efficient as the one that could be initially reached by a monopolist producing both Windows and the complementary media player and internalizing cross-price external effects between producers. At the same time, the fact that such a pricing is achieved just because of Microsoft's entry has the advantage of having left the market decide which are the best characteristics for a media player, which Microsoft functionally cloned only in the "second game". Moreover, and also in case we relax the assumption about "functional cloning", in the second game users will have the possibility of choosing between two existing media players.

6.1.1. Putting Microsoft and RealNetworks in the theoretical framework

Operating systems ("OS", e.g. Windows) and media players ("MP", e.g. Windows Media Player) are clearly complementary products: each and every computer is run using an OS and a media player is present on almost any operating system. For sure, a MP cannot run without an OS and an OS without a MP is considered incomplete by most users. Home users surely want to have a media player, but also business users may be interested in an increasing number of non-entertainment-related multimedia contents (e.g. news). Moreover, even though it is possible to run more than one media player on each OS, typically users have just one favorite MP: this is clearly an approximation, and a quite strong one, however it is at least true that every user can have only one default MP.[256]

Direct and indirect network externalities are very strong in the field of operating systems. If we also consider the crucial importance of strong relationships with hardware producers and hardware compatibility issues, we may be tempted to argue that PC OS market is very near to a natural monopoly characterized by "economies of scale" both on the supply and on the demand side of the market. This view of the OS market may be extreme,[257] but it is not completely unreasonable to treat Microsoft – at least at first approximation – as a monopolist in this market (with a market share constantly over 95% and significant market power according to antitrust authorities on both sides of the Atlantic).[258]

Unlike the OS market, the market for media players (MPs) cannot be considered as an almost natural monopoly. Even if direct network externalities are significant (because of the utility coming from file sharing and similar activities, which is related to the possibility of having compatible MPs), indirect network

[254] COMP/37.792 - MICROSOFT/ W2000. Technical and market related data I quote in this paper come from this source.

[255] At this preliminary stage of the analysis, I assume this time to be sufficient for the complement producer to recoup his sunk cost and make a profit, so that I do not have to deal with any participation constraint for the follower.

[256] A "default media player" is the one the OS is running when the user tries to access a certain media file without explicitly choosing any software to perform this operation.

[257] Someone could argue that Linux OS seems now to be a credible threat to Microsoft's dominance, but – for the moment – this is no more than a potential menace.

[258] See the analysis of relevant antitrust markets in *Microsoft III* and *Microsoft IV* cases. See also, for instance, the analysis of DAVID S. EVANS, *The Antitrust Economics of Two-sided Markets*, Aei-Brookings Joint Center For Regulatory Studies, Related Publication 02-13 (September, 2002), in particular at pp. 17—20.

externalities are less relevant (because these externalities come from the existence of a multitude of complementary products)[259] and relationships with hardware producers are less important.[260] Nevertheless, also in the market for MPs – because of direct network externalities – a unique producer is likely to be in a dominant position at any given time. At the same time, it is likely that a Schumpeterian competition "for the market", rather than "in the market," will put more pressure on MP than on OS producers.

The previous description of the OS and MP markets is compatible with a model describing both Microsoft and RealNetworks as monopolistic producers of two complementary products, but Microsoft seems to be in a relatively stronger and more stable position with respect to RealNetworks. For this reason, I will apply a model very similar to the one of Dari-Mattiacci and Parisi (2006), but with some modifications and I will take into account the role of potential competition for RealNetworks, applying some insights from Dari-Mattiacci and Parisi (2005). Moreover, starting from an intuition provided by Hagiu,[261] this work allows Microsoft to commit to a certain price level for Windows: the basic idea – that I will not expand here – is that long term contracts with Original Equipment Manufactures can be used to credibly fix the price of Windows for significantly long periods. In this way, Microsoft acts like a leader in a Stackelberg duopoly model, while producers of complements (in this case, media players, like RealNetworks for Real One media player) act as followers.[262]

As in Dari-Mattiacci and Parisi (2006), both price-based and quantity-based competition could be analyzed. I argue that in software market quantities can be easily changed, and – at least for operating systems and some very widespread applications – the cost of changing prices can be increased by the need to modify contracts with Original Equipment Manufacturers and other intermediaries in the distribution chain: these elements suggest the use of a Bertrand-like model where firms are choosing prices (as in the "price-setting complementary oligopoly" section of Dari-Mattiacci and Parisi (2006)).

6.1.2. The "First Game": leader/follower price-setting complementary oligopoly

The Stackelberg-like modification of the complementary oligopoly model consists of two stages: in the first one, the leader (a platform producer, like Microsoft, in a position near to natural monopoly) can credibly commit to any price; in the second stage, the follower observes the price fixed by the leader and decides his own price, then the market observes both prices and demand is determined. This simple model can be easily solved by backward induction.

In the last stage of this "first game", the follower (e.g. RealNetworks) will choose its price level exactly as a strategic duopolist in Dari-Mattiacci and Parisi (2006). Notice also that the results of my modified model are – from a quantitative point of view – just the symmetric case of the "Sequential Anticommons" case of the model of Parisi, Schultz and Depoorter (2005).[263] However, among the possible ways of framing the problem, I prefer to extend the – admittedly less general – model of Dari-Mattiacci and Parisi (2006), in order to be able to keep the same notation of the original authors and allow an easy comparison of the results. Moreover, a nice feature of this approach is that – in such a simple setting – it is also possible to directly compare the results of the sequential and non-sequential model.

Let us define the demand for the final good as $P = a - bQ \Leftrightarrow Q = \frac{a-p}{b}$, where $P = p_f + p_l$ is the price of the bundle of the leader's and follower's products (e.g. Microsoft's Windows operating system priced at p_l

[259] But it should be observed that the relationship between media players and digital rights management systems may be such that – in the near future – there will be very significant network externalities associated with the choices of DRM systems. This could have been the real target of Microsoft's monopolization strategy.

[260] One could argue that there are strong linkages between media players, file formats and hardware devices for playing music: this is probably true in some ways, but these relations are different to the ones concerning operating systems and barriers to entry are surely lower in the MP market than in the OS market.

[261] See ANDREI HAGIU, *Two-sided Platforms: Pricing and Social Efficiency*, Harvard Business School and Research Institute of Economy Trade and Industry working paper, Cambridge, Mass. (2005) and ANDREI HAGIU, *Pricing and Commitment by Two-Sided Platforms*, 37 Rand Journal of Economics, 720—737 (2006).

[262] I will call my extension a "Stackelberg-like problem" because the typical Stackelberg model is an extension of a Cournot (quantity-based) and not of a Bertrand-like (price-based) model of competition.

[263] See FRANCESCO PARISI, et al., *Duality in Property: Commons and Anticommons*, 25 International Review of Law and Economics, 578—591 (2005).

and RealNetworks' Real One media player priced at p_f) and Q is the quantity demanded (representing both goods, bought in a 1:1 proportion by all consumers).

For simplicity (and without loss of generality) we assume that marginal costs are zero (which is also a quite realistic assumption in the field of software).[264]

Hence the follower (e.g. RealNetworks) is solving the following problem:

$$\max_{p_f}\left[p_f \cdot Q\right] \Rightarrow \max_{p_f}\left[p_f \cdot \left(a - p_f - p_l\right)/b\right]$$

and the first order condition of the follower's problem yields:[265]

$$\frac{\partial \Pi_f}{\partial p_f} = \frac{a - p_f - p_l}{b} - \frac{p_f}{b} = 0 \Rightarrow p_f = \frac{a - p_l}{2}$$

The leader (e.g. Microsoft) will internalize this reaction function in his maximization problem, so that he is solving:

$$\max_{p_l}\left[p_l \cdot Q\right] \Rightarrow \max_{p_l}\left[p_l \cdot \left(a - \frac{a - p_l}{2} - p_l \right) \Big/ b \right]$$

and the first order condition of the leader is thus:

$$\frac{\partial \Pi_l}{\partial p_l} = a - \frac{a}{2} + p_l - 2p_l = 0 \Rightarrow p_l = \frac{a}{2}$$

If the leader is able to credibly commit to this price ($p_l = \frac{a}{2}$), the follower will chose $p_f = \frac{a}{4}$, so that the total price will be $P = p_f + p_l = \frac{3}{4}a$ and the quantity $Q = \frac{a - 3/4\,a}{b} = \frac{1}{4} \cdot \frac{a}{b}$.

Let me now compare the previous results with those obtained by Dari-Mattiacci and Parisi (2006). The authors analyzed a model identical to the previous one, but in the two cases of a unique monopoly producing both complementary goods (equivalent to the case of joint perfectly collusive control of the market) and of a duopolistic industry (with two symmetric players, i.e. without a leader with the possibility of committing himself in a preliminary stage of the game):

Comparative table	Monopoly (joint control)	Duopoly (standard "Anticommons")	Stackelberg-like model (leader/follower)
Total price	$P_M = \frac{a}{2}$	$P_A = \frac{a}{3} + \frac{a}{3} = \frac{2}{3}a$	$P_S = \frac{a}{2} + \frac{a}{4} = \frac{3}{4}a$
Quantity produced	$Q_M = \frac{1}{2} \cdot \frac{a}{b}$	$Q_A = \frac{1}{3} \cdot \frac{a}{b}$	$Q_S = \frac{1}{4} \cdot \frac{a}{b}$
Total profits	$\Pi_M = \frac{1}{4} \cdot \frac{a^2}{b}$	$\Pi_A = \frac{2}{9} \cdot \frac{a^2}{b}$	$\Pi_S = \frac{3}{16} \cdot \frac{a^2}{b}$

From the results in the previous table it is evident that $\Pi_M > \Pi_A > \Pi_S$ as one could expect, but $P_M < P_A < P_S$ and $Q_M > Q_A > Q_S$. This implies that – even if the industry as a whole would obtain more profits in the monopolistic case – also the level of social welfare (which is related to the quantity produced) would be higher in the case of a monopolistic (or perfectly collusive) control of the two complements.

[264] The marginal cost is close to zero, for instance, if the distribution takes place through the Internet.

[265] This condition is also sufficient because of the concavity of objective function.

From the point of view of social welfare, the outcome of the Stackelberg-like model is also worse than the standard (duopoly) Anticommons case discussed by Dari-Mattiacci and Parisi (2006), even if – as one could expect – the leader's profit is higher than the profit of a single duopolist. Clearly, the strategic commitment of the leader is such that the follower is forced to choose a lower price than in the duopoly case. Indeed, $p_f = \frac{a}{4} = \frac{1}{2}P_M$, so that we can say that the leader is able to force the follower to price at the optimal level from the point of view of the maximization of joint profits, but the follower's lower price is more than compensated for by the higher price chosen by the leader himself. In other words, the leader can force the follower to (*de facto*) internalize the effect of an excessive p_f on Π_l, but the leader himself is completely ignoring the negative effect of p_l on Π_f (choosing an high p_l is actually the instrument used by the leader to induce the follower to reduce p_f). In other words, the strategic position of the leader allows it to "force" the other player to set a price, which is nearer to the optimal one for the leader itself, but which is also preferable for society: the problem is that this improvement is more than offset by the leader's increase in price.

It's also interesting to notice that $P_M = \frac{a}{2} = p_l$, so that – once the price of the leader has been fixed – the only way to reach "efficiency" would be to price at zero the complementary good produced by the follower. This result will have an interesting implication in the following part of this section. Before going on, notice that I called "efficient" the solution chosen by a unique controller of the two goods, internalizing all external effects between producers. Clearly, the efficiency of this solution may be questioned, since the first best solution, from the point of view of static efficiency in society, would be $P=MC=0$. Notice, however, that this would imply zero profits and no incentives to innovate because of the existence of sunk costs. Hence – assuming that monopoly is considered by society an efficient way to finance the creation of immaterial goods (as it seems to be the case, given the existence of intellectual property) – it may surely be argued that the purely monopolistic solution is more efficient than other solutions not internalizing cross price externalities.[266]

6.1.3. The "Second Game": functional copy and bundling

At some time after the creation of the complementary good, I assume that potential entrants (including the leader himself) become able to produce a functional copy of the complementary good initially developed by the follower.[267] In order to be allowed to treat this situation as a completely different game with respect to the previous one, here I assume that there are no binding participation constraints related to the participation of the follower to the first game. In other words, I assume that the lead-time of the first developer of the complementary product (RealNetworks, the "follower", producing Real One MP) is sufficient for him to recoup fixed costs and earn a sufficient level of profit to induce innovation in the first place.[268] Alternatively, it may be sufficient to assume that the production of the complementary good is, in any case, profitable for its producer (for instance, because it allows reaping profits in other markets, e.g. selling music over the Internet in the case of RealNetworks).

Please, notice that describing this "second game" I will continue to call "leader" the Stackelberg-leader of the "first game" (Microsoft) and "follower" the producer of the complementary product (RealNetworks). However, the "follower" is also the "incumbent" or "first-comer" in the market of his complementary product (Real One was the "first-comer" producer in the field of medial players, even though it player the role of follower in the Stackelberg-like model above).

---

[266] Again, this is the same principle for which a vertical integrated monopoly – or an agreement between a monopolistic producer and distributor – may be preferred to two independent monopolists practicing double marginalization.

[267] See above, § *4.1. Functional clones and late comers.*

[268] Clearly, the model could be extended to explicitly take into account participation constraints in the "first game" on the basis of the solution of the "second game": in this case we would have a unique game solved by backward induction and the leader (Microsoft) could use its power to commit to a certain price of Windows (in the first stage of the first game) in order to fine tune the follower's level of profits (during the first game and the period between the two games), so that he meets his participation constraint (this is the case, if the participation of the follower is sufficiently beneficial for the level of leader's profits, as it's usually the case if the follower is more efficient than the leader in performing the initial development of the complementary product).

So, I assume that – after some time – a functional copy of the complementary good, initially developed by the follower, can be realized and that this potentially leads to a new (independent[269]) game. The creation of this functional copy requires the undertaking of a significant sunk investment,[270] basically in order to hire programmers: using the wording of the second paper, this is more an investment in "development" than in "research" (obviously, in reality, both kind of investments may be performed). This investment produces a good that is technically as performant as the original one and this technical outcome is subject to a very low level of uncertainty. Moreover, the functional copy is not infringing intellectual property rights of the follower (here I assume that the good is protected only by copyright and not by patents), because the source code is completely re-written and only ideas, methods of operation and other functional concepts (not protected by copyright) are reproduced. The only limit of the functional copy may be that – for the reasons discussed, in particular, in the second paper – it may not be 100% compatible with file formats and other technical specifications of the incumbent product (however, a substantial degree of compatibility would be normally possible). In other words, a copy of the functionally copied program is, *ceteris paribus*, as good as the original one. However, other things are not equal and minor incompatibilities imply that any late comer in the complementary market may enjoy lower network effects (both direct and indirect), in an environment dominated by copies of the original version (produced by the "follower" of the first model, e. g. Real Networks).

As I tried to show,[271] the previous assumption ("functional copies of copyrighted programs are easy to create") is not particularly unrealistic; however, one could consider my argument as falsified by the absence of very frequent new entries in software markets. I suggest that the reason why new entrants may decide not to fight against an incumbent are – at least – two: first of all, with zero marginal cost and very flexible production processes, (Bertrand) competition between perfect substitutes product would easily drive prices (and profits) to zero; second, the incumbent has some lead-time and this guarantees him a competitive advantage coming, among other things, from direct (and eventually indirect[272]) network effects. Hence, to represent a competitive threat, any new-comer would need to subsidize his first consumers in some way in order to convince them to leave a technically equivalent software, which is used by more people. However, this would imply that the cost of late comers will be higher than the costs of incumbents, hence entrance with functional cloning will not happen. Briefly, the combined effect of copyright law (which is sufficient to avoid complete free riding and requires any "functional copier" to undertake relevant sunk costs) and high network effects developed during the lead-time of the first-comer is sufficient to avoid massive entry and guarantees to the first-comer (e.g. RealNetworks) the possibility of recouping some investments.[273]

But the previous reasoning did not take into account the peculiar position of the platform controller. Indeed, for the leader of the first Stackelberg-like game (Microsoft), the entry of a competitor would be very beneficial: in fact, the existence of a single substitute to the complementary product (e.g. RealOne media player) would be sufficient – in a Bertrand-like price competition setting – to drive to zero the price of the complementary good.[274] And, looking at the results of the "Comparative table",[275] lowering to zero the price of the complementary product would increase the profit of the leader (e. g. Microsoft) from $\Pi_l = \frac{1}{8} \cdot \frac{a^2}{b}$ to

[269] See the previous footnote for a preliminary discussion of the case in which the two games cannot be considered as independent.

[270] We may assume that the amount of resources needed to create this functional copy is equivalent or only slightly lower to that needed to develop the original version of the complement, but now there is no uncertainty concerning the existence of a relevant demand for this kind of software.

[271] See above, § *4.1. Functional clones and late comers.*

[272] This is the case if some add-ons (complementary products), specifically designed to work with the incumbent's software, have been realised: a functional copy of the incumbent's product – even if it's a perfect substitute for final users – may not be compatible for technical reasons with these products (especially if the incumbent used in a strategic way a combination of trade secret and non-disclosure agreements with the developers of these add-on components).

[273] For simplicity, I do not consider here the possibility that some new entrants are (or think to be) able to produce a much better product with respect to that of the incumbent.

[274] For the discussion of more realistic and complex cases than basic Bertrand competition with undifferentiated goods, see the model of Dari-Mattiacci and Parisi (2005), where they answer to the following question: "*how many substitutes for each complement are necessary to render the presence of multiple sellers preferable to monopoly?*". The authors show that two substitutes per component (complement) are normally sufficient. In other words, a single competitor for any producer of complements is enough to drive prices down to a level which is preferable to the monopolistic one.

[275] I assume that the commitment of the leader concerning his price is still holding in this second game.

$\Pi_M = \frac{1}{4} \cdot \frac{a^2}{b}$. This imply that the leader (and only the leader, not any potential entrant) is willing to invest up to $\Pi_M - \Pi_l = \frac{1}{8} \cdot \frac{a^2}{b}$ in order to develop (or buy from third parties) a functional copy of the follower's (e.g. Real Network's) complementary good and to overcome network effects protecting this first-comer's software.

Hence, the leader has an incentive to create a substitute for the follower's product and give it away for free, just to reduce the follower's price (all that always assuming that the follower is still be able to remain in the market and innovate, collecting revenues otherwise: in my simplified model, this is guaranteed by the fact that the participation constraint of the follower is satisfied during the first stage of the game). Moreover, in the real world, the leader also has a powerful device to overcome the barrier to entry represented by network effects enjoyed by the incumbent at a relative low cost. Tying (or simple bundling, when the two products are one as good as the other) may be a way to do so.

As broadly discussed, the controller of the OS (or similar platform) has the possibility of bundling a functional copy of the complementary product to the OS itself, so that users may already have it installed when they purchase a brand new PC. However, under the principle of modularity, advanced users and/or OEMs may easily decide to uninstall the bundled product and install that of the follower (RealNetworks'). And they will do so, unless the leader's product is technically superior and/or differentiated enough to present sufficient advantages (at least for some users) in order to overcome the network effects of the complementary market incumbent. In any case, RealNetwork's pricing policies will be constrained: before entry, it had the opportunity of fully exercising its market power; after, it cannot price at more than the value users attach to the network effects that will be lost shifting from its product to that of Microsoft (assuming that the two products are technically equivalent).[276] In my simplified model, the price of both products would drop to zero, since network effects are neglected, but also in the real world the leader could significantly constrain the capacity of RealNetwork to exercise market power through prices in this complementary market, and this would reduce inefficiencies coming from cross price externalities (i.e. "double marginalization-like" market failures).

In any case, in this setting Microsoft would have no interest in excluding RealNetworks with a technically inferior product. Moreover, in reality both producers could start an "innovation war", as happened when Microsoft tried (successfully) to conquer the browser market, fighting against Netscape. In the model, Microsoft would not have a special incentive to use particular bundling and pricing strategies in order to foreclose RealNetworks, once its price drops to zero. However, in the real world, when some of the exceptions to the ICE paradigm are in place, Microsoft could be temped to use other tools to "cut off the air [supply]" of potential competitors.[277] That may happen when controlling the complementary market offers a competitive advantage in a third market (like the one of DRM-protected contents, once one control media players). That is even more likely in cases where controlling the complementary market – like that of workgroup servers– may allow the extraction of more surplus from Windows consumers with a higher willingness-to-pay (as business users), using a second degree price discrimination strategy.[278] Moreover, and this is probably the most worrying case, it is possible that shaping the complementary market in a certain way (possibly a way that is not in the best interest of consumers) may protect the monopoly of the platform incumbent. For instance, Microsoft probably thought that the browser could represent a crucial platform for internet users – potentially able to "commoditize" operating systems (and such a possibility may still exist and actually push Google to invest in a browser such as its Chrome). This is probably why Microsoft decided to crush Netscape's Navigator browser as fast as possible. However, a similar threat coming from workgroup servers operating systems is even clearer, in a world were computers are more and more part of an interconnected world and not simply stand-alone machines. Hence, the fear of potential competition could have pushed Microsoft to disfavor competing workgroup server producers.

---

[276] Actually, adding additional possibilities of strategic interaction in this "game", RealNetworks may decide to price its RealPlayer low enough as to avoid Microsoft's entry at all. That, unless Microsoft considers that its entry may be beneficial in terms of quality, increasing the overall value of the system (platform+complementary good) also thanks to competitive pressure to innovate further on RealNetworks.

[277] See above § *2.1.1. The ICE paradigm.*

[278] See KÜHN & REENEN, *Interoperability and Market Foreclosure*, p. 20.

What is important is that this small model showed that forcing RealNetworks to price at zero may be both in the interest of Microsoft and socially beneficial. Hence, it is fortunate that the Commission Decision did not preclude this possibility. At the same time, as the previous parts of the paper showed, also the fact of preventing Microsoft from making a strategic use of exclusionary bundling – with a de facto negative price of WMP – is likely to prevent abuses. Indeed, zero is also the (industrial)[279] marginal cost of a copy of WMP, hence this is a quite natural threshold as a minimum price for this product.

7. Notes and open issues

Applying the principles of modularity and disclosure as suggested in this paper may, in my opinion, reduce the need for the future intervention on the part of competition authorities in software markets. Indeed, the likelihood of these interventions is, at the moment, very high: the next chapter of Microsoft's saga is already being written in Brussels. In any case, there are several unsettled issues in the optimal competition policy to adopt toward incumbents such as Microsoft, and clarity about some topics, such as the tying of new functionalities to Windows, may help in focusing the attention of the Commission on other possible abuses.

7.1. Microsoft V: the next chapter

A major Microsoft V[280] antitrust case may be forthcoming and the European Commission has already started a new formal procedure. This last wave of investigations has been triggered by a complaint by Opera Software,[281] but some of the issues at hand are as old as Windows 95 and are now revived by a recent surge of competition in the browser market (of course, this competition mainly concerns advanced users being aware of existing alternatives to the standard browser coming bundled with Windows).

The crucial element of the complaint concerns the tying of Internet Explorer (IE) to Windows (requiring the existence of unbundled versions and/or a must-carry order). Additionally, Opera is asking the Commission to force Microsoft to "follow fundamental and open Web standards accepted by the Web-authoring communities."[282] In fact, according to Opera, "Microsoft's unilateral control over standards in some markets creates a *de facto* standard that is more costly to support, harder to maintain, and technologically inferior and that can even expose users to security risks."[283]

This may be an empirical example of the fact that bundling without information withholding – indeed, using secrecy in order to protect these standards is almost impossible, because they have to be disclosed to website developers – may have some, but not spectacularly severe, anticompetitive effects. In fact, competitors remain free to implement – even though with some delay – Microsoft's proprietary standards. Actually, pending these strategies of Microsoft, Mozilla's Firefox reached a market of about 20%.[284] New competitors

---

[279] As I already mentioned, a copy of WMP may actually have a negative cost, in the sense that it creates the possibility of exploiting network externalities and/or similar revenues, as the one coming from advertising.

[280] I numbered this case "Microsoft V" adopting a western-centric approach. In fact, Microsoft has been subject to antitrust procedures in various parts of the world. In particular, in South Korea (an impressively growing market for software and any informatics related product), where the Fair Trade Commission fined it of $32 million in December 2005. In that case, the software house had been ordered to unbundle instant messaging, media players and other services from Windows. In response to the case, Microsoft even menaced to quit the Korean market, but never applied this threat. *See* BBC News, *South Korea fines Microsoft $32m*, December 7, 2005, (http://news.bbc.co.uk/2/hi/business/4505698.stm; last visited, June 22, 2008). After that the Seoul High Court rejected Microsoft's stay request in July 2006, the company started to comply with the decision, of the FTC, starting to selling an unbundled version of Windows (stripped-down of media player or instant messaging software) in the Korean market. *See* Microsoft's press release *Microsoft Statement Regarding the Korean Court's Decision*, July 4, 2006 (http://www.microsoft.com/presspass/legal/07-04-06KoreaStatement.mspx; last visited, June 22, 2008); John Oates, *Microsoft falls into line with Korean anti-trust order*, The Register, August 23, 2006 (http://www.theregister.co.uk/2006/08/23/ms_korea_xp/; last visited, June 22, 2008); John Oates, *Microsoft drops South Korea anti-trust appeal*, The Register, October 16, 2007 (http://www.theregister.co.uk/2007/10/16/ms_appeals_south_korea/; last visited, June 22, 2008).

[281] Opera is an innovative Norwegian software house, active in the production of a browser with the same name, which introduced, in this market, innovative features like tabbed browsing (then introduced also by Mozilla's Firefox, Microsoft's Internet Explorer and – lately – by Google's Chrome), integrated search bars and mouse gestures. See Opera's press release *Opera files antitrust complaint with the EU*, December 13, 2007(available at http://www.opera.com/pressreleases/en/2007/12/13/; last visited June 22, 2008).

[282] *Ibidem.*

[283] *Ibidem.*

[284] See the second paper of the dissertation at hand, footnote 10.

also entered the market, notably Google with its Chrome browser.[285] In any case, according to the approach of the paper at hand, in order to further reduce anti-competitive effects it could be necessary to force Microsoft to perform various steps. A straightforward application of the principle of modularity would likely be beneficial for consumers, but Microsoft could credibly raise several efficiency justifications for this specific bundling, because various parts of Windows may share code with IE. Despite that, it would be possible to allow the disinstallation of – at least – the user interface of Internet Explorer and to verify that users are not forced to use IE when it is not technically necessary. In any case, if Microsoft decided to defend its bundling (as it is very likely), it should timely disclose[286] to competitors the specifications of non-public standards that will be incorporated into new versions of Internet Explorer, so that they may decide if they want to implement these standards as well (allowing competition "inside the standard", in exchange for the possibility of bundling). Moreover, all APIs used by IE to interoperate with Windows should be disclosed by Microsoft, as well as APIs exposed by IE itself.

That having been said, forcing Microsoft to respect public standards would go much further than my recommendations (even though I do not exclude that there may be good reasons to go further with respect to what has been discussed: indeed, I intentionally limited my recommendation to quite unambiguously welfare enhancing policies). Indeed, this paper – or any scholarly work of which I am aware – did not clearly show that standards favored by a single private undertaking are necessarily worse than publicly established standards. Indeed, some coordination problems may suggest that, in principle, the opposite may be true. This is why, in order not to loose precious time (because, in software markets in particular, time is money and the amount possessed by Microsoft competitors is far from infinite), the Commission should quickly apply to this case the general principles, which can be derived from the Microsoft IV case,[287] and then take the time necessary to investigate further issues, such as the potential duty of a dominant incumbent to respect public standards and/or to refrain from so-called Embrace-Extend-Extinguish strategies.[288] In other words, the Commission should be urged to "unbundle" some of Microsoft's violations, instead, one may fear that a unique, long investigation is going to be started. For the moment, the European Commission has decided to initiate two distinct formal antitrust investigations against Microsoft, one based (*inter alia*)[289] on Opera's complaint and another one "in the field of interoperability in relation to a complaint by the European Committee for Interoperable Systems (ECIS)."[290]

> In the complaint by ECIS, Microsoft is alleged to have illegally refused to disclose interoperability information across a broad range of products, including information related to its Office suite, a number of its server products, and also in relation to the so called .NET Framework. The Commission's examination will therefore focus on all these areas, including the question whether Microsoft's new file format Office Open XML, as implemented in Office, is sufficiently interoperable with competitors' products.[291]

Unfortunately, the approach followed by the Commission's in its *Microsoft Decision* may require another years-long procedure, ending up with a some-hundred-pages story about Microsoft's violations, which may be convincing enough to pass the scrutiny of European Courts, but which risks not being sufficiently clear as to create unambiguous precedents for the software industry.

---

[285] Google launched its browser at the beginning of September 2008. For more information, see the Wikipedia page http://en.wikipedia.org/wiki/Google_Chrome (last visited September 14, 2008).

[286] For instance, a disclosure taking place when Microsoft's implementation are in the alpha stage of development could be timely.

[287] As Jason Hoida, Deputy General Counsel, Opera puts it: "The European Court of First Instance confirmed in September [2007] that Microsoft has illegally tied Windows Media Player to Windows. We are simply asking the Commission to apply these same, clear principles to the Internet Explorer tie, a tie that has even more profound effects on consumers and innovation." (See *supra* note 281.)

[288] See the US DoJ document *U.S. v. Microsoft: Proposed Findings of Fact*: (available at http://www.usdoj.gov/atr/cases/f2600/2613.htm; last visited July 20, 2008). According to the testimonies collected by the US DoJ, "Paul Maritz also explained to Intel representatives that Microsoft's response to the browser threat was to 'embrace, extend, extinguish'; in other words, Microsoft planned to 'embrace' existing Internet standards, 'extend' them in incompatible ways, and thereby 'extinguish' competitors." For a quick reference, see the Wikipedia (http://en.wikipedia.org/wiki/Embrace,_extend_and_extinguish; last visited July 20, 2008).

[289] "In addition, allegations of tying of other separate software products by Microsoft, including desktop search and Windows Live have been brought to the Commission's attention." *See* European Commission's MEMO/08/19: *Antitrust: Commission initiates formal investigations against Microsoft in two cases of suspected abuse of dominant market position*; Brussels, 14th January 2008.

[290] *Ibidem.*

[291] *Ibidem.*

8. Conclusion

In cases in which a platform owner acquires a quasi-monopolistic position in software markets, it is possible that not only competition "in the market", but also the so-called Schumpeterian competition "for the market" becomes very unlikely, so that this monopoly may become almost permanent or at least very persistent. This is more likely when network effects and learning costs (with the consequent lock-in of users) combine with the ability of the platform controller of evolving its platform to "resist" reverse engineering (making also "self-help" from reverse engineering a very imperfect tool to recreate some competition "inside" the *de facto* standard imposed by the market leader). Indeed, the possibility of defending quasi-monopolistic positions becomes more likely when the platform owner uses a combination of tying and information withholding strategies to eliminate competitive threats and even enlarge its market power in selected adjacent markets. As the ICE paradigm shows, this is especially probable when these complementary markets both offer lucrative opportunities and represent a potential competitive threat for the platform incumbent, if controlled by competitors. In fact, all the anticompetitive strategies of the incumbent are made possible by the fact that intellectual property protection of copyright, while usually sufficient to provide incentives to innovate and flexible enough to permit dynamic innovative markets, cannot exclude the possibility that incumbents with 'deep pockets' may create functional clones of "dangerous complementary products". Despite the fact that these "functional clones" are normally not a profitable product in themselves, platform controllers may use them both as tools to increase competitive pressure on complementary market's controllers (a pro-competitive use, discussed in § 6) and as anti-competitive weapons to eliminate potential competition. In principle, one may argue that an intellectual property setting, forbidding the creation of functional clones, could prevent some anti-competitive strategies of big incumbents. However, the flexibility and relative weakness of copyright protection of software is, in general, more pro- than anti-competitive, as the first two papers of this dissertation tried to show. This is why, in this third paper, I suggest the use of antitrust intervention to avoid some of the most dangerous abuses of (super-)dominant platform controllers, instead of proposing to increase the intellectual property protection afforded to innovative creators of complementary products.

In fact, this paper showed that simple bundling – the fact of selling complementary software along with the core platform – is likely to have just limited anticompetitive effects, especially if not only users, but also original equipment manufacturers (and similar intermediaries, working as consultant for inexperienced users) are free to remove the pieces of software preferred by the platform owner and substitute them with complementary products of their choice. As the literature on tying showed, some exclusionary strategies may still be based on simple bundling (see, for instance, Nalebuff's analysis of exclusionary bundling).[292] However, these strategies are largely prevented by rules imposing that unbundled versions of the platform cannot be cheaper than the bundle. Hence, the approach that has been followed by the European Commission in the Microsoft case, is far from being completely unbinding and useless. To the opposite, this paper discussed how the rules imposed by the Commission may both be binding and have some efficient properties, in so far as they may allow the platform controller to reduce inefficiencies coming from cross price externalities among complementary products. Of course, the limit of the approach of the Commission is that it has been imposed *ex post* and *una tantum*, without coupling it with appropriate remedies in order to restore competition in the market for media players.

Also information withholding alone is not normally in the interest of software producers, not even if they are dominant. In fact, information withholding may exclude (actual or potential) competitors, but it also imposes severe costs on other producers of complementary products, the existence of which is beneficial for the overall value of the hardware/software system, at the centre of which a dominant platform is able to extract a significant share of this generated surplus. For instance, Microsoft, while it was just one of the competitors in the workgroup server operating systems market, had a relatively open approach: it started withholding interoperability information only when the strategy had been coupled with the tying of some client-side parts of its server technology to Windows client operating systems. In fact, information withholding is complementary to tying in so far as it avoids competition "inside" the *de facto* standard that tying imposes.

In any case, and despite the fact that non-disclosure of interoperability information has its maximum anti-competitive effects when coupled with tying, when a platform is really super-dominant, some selective

[292] NALEBUFF, *Exclusionary Bundling*.

withholding of information may allow it to remain in this position in a more stable way. Hence, the paper recommended that – just for dominant positions "verging on monopoly" (i.e. in case of super-dominance) – an obligation to disclose/license interoperability information may be imposed. The paper also offered some arguments in favor of an obligation of free disclosure, however I admitted that this position may not always be tenable, hence I discussed also the possibility of RAND licensing terms. In any case, this disclosure obligation should be carefully (and quickly) enforced especially when complementary products need interoperability information, with particular care in cases where an exception to the ICE paradigm of Farrell and Weiser may be in place.

Focusing on cases in which new functionalities are added to an existing super-dominant platform, I discussed the main policy recommendation of the paper at hand. In these cases, a general principle of modularity should guide the intervention of competition policy. As a general rule, no tying of functions which could, technologically speaking, be considered separate should be allowed. However, the evaluation of what needs to be tied to the platform for technological reasons may be difficult, if not impossible, to perform for antitrust agencies. Hence, an alternative may be proposed to super-dominant incumbents. If they evaluate that technological tying is necessary (i.e. a bundling creating a physically indivisible product), the disclosure obligation associated to the super-dominance on the platform should be extended to the bundled applications. This disclosure should encompass both application programming interfaces and communication protocols that the tied applications expose and APIs they use to interoperate with the rest of the platform (this disclosure should encompass also file formats, including DRM-related protections, used by the tied product). Moreover, modularity should be preserved, where there is no cost in doing so: for instance, it should be possible to remove at least the user interface of pieces of software which are substituted by alternative products (e.g. the users interface of Internet Explorer, if I install Mozilla's Firefox). And this freedom should be granted both to users and to their *de facto* advisors, i.e. OEMs and other intermediaries.

When modularity is chosen, instead of disclosure, unbundled versions must be available to OEMs and to users. As far as prices are concerned, the only limitation may be related to the fact that the unbundled version may not be more costly than the bundled one. Of course, this limitation implies that the price for finding the unbundled version must be reasonable; hence, even if distributors do not seem to demand it, it should be available through e-commerce circuits or the like.[293]

Notice also that, if the European Commission wants to make clarity in this field, interoperability should be clearly described as a tool of the Commission's industrial policy and not just as a goal worth a small exception to the general rule of inviolability of intellectual property protection (as it may seem to be in article 6 of the Software Directive). Moreover, in the field of software, interoperability and modularity could be considered as principles of standing, similar to that of intellectual property and surely as principles as important as trade secret. In fact, all these principles should be seen as instrumental to the achievement of innovation. Hence, in any case, no super-dominant undertaking should be allowed to violate, at the same time, the principles of modularity and interoperability.

Whatever principle (among modularity and interoperability/disclosure) super-dominant firms may decide to follow, there may be some advantages (and some residual risks for competition). On the one hand, the likely result is that a firm will opt for technological tying (and implicitly, even if likely unwillingly, accept disclosure) when all potential customers of the platform have a significant demand for the bundled complement and when it is genuinely convinced of the efficiency of technological tying. Indeed, in that case joint production is likely to be efficient – because the platform has an high incentive to increase the total value of the system – and disclosure looks like a reasonable part of the special responsibility of a (super-)dominant firm. Competitors will still be able to exist, but they will likely have to offer differentiated or niche products and/or to compete "within the standard" sponsored by the platform. On the other hand, when the dominant producer of a platform wants to enter another market, which is complementary, but it is not of general interest for all platform users (e.g. Microsoft wanting to enter the workgroup server market), then I expect the dominant firm to opt for modularity. In that case, pieces of software needed by the new product to interoperate with the dominant platform will have to be distributed separately (even if possibly for free), and that will put all competitors on the same ground. Obviously, OEM manufacturers, systems administrators

[293] Other detailed recommendations, which I cannot fully discuss here, may be appropriate: simply as an example, it would be appropriate also to let people receiving their copy of the platform preinstalled on a computer to chose a simple option to "strip down" the platform of all technically unnecessary complementary products. (About that, let me just mention that also the *de facto* impossibility of buying personal computers without a copy of Microsoft's Windows preinstalled on them would require some antitrust scrutiny.)

and other agents able to influence the initial configuration of computers carrying the dominant platform will be free to install any of these pieces of software (i.e. both from Microsoft and its competitors); however, Microsoft will not be able to leverage its Windows Monopoly power to do so. In particular, it will not be possible to make discounts on Windows conditional to the installation of these pieces of software. Finally, antitrust agencies will hopefully verify if any other discount and/or incentive strategy of Microsoft may be considered as predation, without any possibility of recoupment, unless the strengthening or maintainment of the original platform monopoly is considered.

Frankly, I do not think that particular limitations to the general principle of interoperability/disclosure is necessary: notice that I already recommended to limit its scope stressing that just super-dominant undertakings should respect it, as a consequence of their peculiar "special responsibility" and just for their super-dominant platform and/or for technologically tied complementary products. However, as an aside, notice that, if one considers the aforementioned approach as being excessively interventionist, the disclosure duty may be limited to cases in which one of the exceptions to the ICE paradigm is clearly in place. Since some exception may always be considered to be in place, one may restrict the field to the clearest exceptions and mandate modularity and interoperability only with respect to some precise categories of middleware representing a potential competitive threat (and similar programs exposing communication protocols). In any case, Microsoft cases both in the US and in the EU concerned similar products (indeed, nobody ever cared about the fact that Notepad or Paint are bundled with Windows).

Overall, the European Microsoft Decision applied principles, which are compatible – even if not exactly identical – to the ones recommended in this paper. First of all, in the part of the Decision concerning Windows Media Player, Microsoft has been prevented from practicing the "technological tying" of a piece of software exposing proprietary APIs and using proprietary file formats, including Microsoft's digital rights management technology. At the same time, the software house has been allowed to compete in the most aggressive way (including practicing a zero price) with producers of complements. The Commission did not offer Microsoft an alternative to unbundling in the form of mandatory disclosure of its secret media player related information, but – in this case – one may guess that Microsoft would not have taken up this possibility, arguably being far from ready to adopt an "open DRM" approach to the distribution of technologically protected multimedia contents. About this part of the Decision, I just criticize, following Ayres and Nalebuff, the fact that a must-carry obligation should have been imposed, as an additional remedy to restore competition (and just because of the fact that the unbundling order had been imposed with years of delay).

Also as far as the workgroup server OS market is concerned, the Commission Decision may be considered largely coherent with what I proposed. Indeed, Microsoft clearly decided to bundle new server-related functionalities in client PC, hence it had to face a disclosure order. However, if competition policy intervention had been quicker (and base on clearer principles), it would have been possible to offer to Microsoft an alternative between mandatory disclosure and unbundling of network features from Windows clients (at least as long as the software house is not super-dominant in the server operating systems market as well,[294] something that – in principle – could never happen, especially if it had been prevented from coupling tying and information withholding to conquer this market). As I discussed, the part of Microsoft's disclosure obligation concerning server-to-server interoperability cannot be fully justified under the approach proposed in the paper at hand, however one could recommend it, again, as an additional measure to restore competition (in this case, the measure should likely have been limited in time, for instance to five or ten years).

Notice also that the simple model presented in this paper can be used to reply to the critiques of several commentators[295] saying that the European Commission adopted a wrong (and/or useless) decision in allowing Microsoft to price at zero his Windows Media Player. In fact, taking into account complementarities

[294] Here, the reference to the "server operating systems market", instead of the "workgroup server operating systems market" is not an accident. Once the competitive advantage of Microsoft over the interaction with clients is removed, various sever operating systems producers may compete with Microsoft on a more equal ground, in this way potential competition on the workgroup segment from other segments of the server operating systems market may be significant (so that we may actually face a unique market).

[295] For instance, the website www.thestandard.com (and other websites) quoted an anonymous commentator (working for the EU Commission), saying that: "If the two versions [i.e. Windows XP with WMP and Windows XP N] were sold at the same price then obviously it would be less hassle for PC makers simply to continue selling the bundled version they have been selling until now". Several newspapers (and some scholars) published similar comments. Indeed, it appears that a very limited number of copies of Windows XP N has been sold: see footnote 232.

between Windows and Windows Media Player and the fact that Microsoft committed to the price of Windows before the wide spreading of MPs (as in the Stackelberg-like model I proposed), it is arguable that a zero-price strategy is perfectly rational, non-predatory (marginal costs are zero) and also beneficial for social welfare.

To conclude, the *Commission's Microsoft Decision* already sets an important precedent for future bundling strategies of dominant firms, stressing the importance of the freedom of choice of original equipment manufacturers as "consultants" of less experienced final consumers. It is important to interpret the Decision as a precedent stating that modularity is a principle that super-dominant firms must respect in software markets. However, the Decision seems to provide far less clarity in the field of information withholding, where the actual intervention of the Commission may be appropriate, but its analysis, including the so-called "incentive balancing test" are far from providing the needed degree of legal certainty. Moreover, the Decision seems to have largely missed the point concerning the complementarity between tying and information withholding (despite the fact that two investigations concerning this kind of practices has been merged and decided in two chapter of the same Decision, so showing at least some awareness about the existence of a unique complex anticompetitive strategy of Microsoft). Integrating into the Commission's competition policy a higher degree of awareness about this complementarity may increase the flexibility of the Commission's intervention in software markets. Overall, the Commission's approach would also still benefit from more clarity and administrability, which could be achieved, for instance, issuing guidelines for software markets, based on a proposal like the one described in this paper.

Bibliography

C. AHLBORN, et al., *The Antitrust Economics of Tying: A Farewell to Per Se Illegality*, The Antitrust Bulletin (2003)
PHILIP AREEDA, *Essential Facilities: An Epithet in Need of Limiting Principles*, 58 Antitrust L.J., 841 (1989)
IAN AYRES & BARRY NALEBUFF, *Going Soft on Microsoft? The EU's Antitrust Case and Remedy*, 2 The Economists' Voice, Article 4 (2005)
JONATHAN BAND & MASANOBU KATOH, *Interfaces on Trial -- Intellectual Property and Interoperability in the Global Software Industy*, (Jonathan Band ed., Westview Press First ed, Boulder, Colorado. 1995)
LUÍS CABRAL, *Economia Industriale*, (Carocci, Rome. 2002)
D. W. CARLTON & M. WALDMAN, *The Strategic Use of Tying to Preserve and Create Market Power in Evolving Industries*, 33 The Rand Journal of Economics, 194--220 (2002)
DENNIS W. CARLTON & MICHAEL WALDMAN, *Tying, Upgrades, and Switching Costs in Durable-Goods Markets*, NBER working paper series, n. 11407 (June, 2005)
GIUSEPPE DARI-MATTIACCI & FRANCESCO PARISI, *Substituting Complements*, 2 Journal of Competition Law and Economics, 333--347 (2006)
BEN DEPOORTER & FRANCESCO PARISI, *The Market for Intellectual Property: The Case of Complementary Oligopoly*, *in* The Economics of Copyright: Developments in Research and Analysis (W. Gordon & R. Watt eds., 2003)
JOSEF DREXL, *IMS Health and Trinko - Antitrust Placebo for Consumers Instead of Sound Economics in Refusal-to-Deal Cases*, 35 International Review of Intellectual Property and Competition Law, 788--808 (2004)
IAN EAGLES & LOUISE LONGDIN, *Microsoft's Refusal to Disclose Software Interoperability Information and the Court of First Instance*, 30 European Intellectual Property Review, 205--208 (2008)
DAVID S. EVANS, *The Antitrust Economics of Two-sided Markets*, Aei-Brookings Joint Center For Regulatory Studies, Related Publication 02-13 (September, 2002)
DAVID S. EVANS, et al., *Invisible Engines -- How Software Platforms Drive Innovation and Transform Industries*, (David S. Evans ed., MIT Press First paperback ed. 2008)
DAVID S. EVANS, et al., *Did Microsoft Harm Consumers? Two Opposing Views*, (David S. Evans ed., AEI Press. 2000)
JOSEPH FARRELL & PHILIP J. WEISER, *Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age*, 17 Harvard Journal of Law & Technology, 85 (2003)
HARRY FIRST, *Netscape is Dead: Remedy Lessons from the Microsoft Litigation*, New York University - School of Law working paper (August, 2008)
HARRY FIRST, *Strong Spine, Weak Underbelly: The CFI Microsoft Decision*, NYU Law and Economics Research Paper No. 08-17 (April, 2008)
N. GANDAL, et al., *Ain't it 'Suite'? Strategic Bundling in the PC Office Software Market*, mimeo, Columbia University (2004)
ANDREI HAGIU, *Two-sided Platforms: Pricing and Social Efficiency*, Harvard Business School and Research Institute of Economy Trade and Industry working paper, Cambridge, Mass. (2005)
ANDREI HAGIU, *Pricing and Commitment by Two-Sided Platforms*, 37 Rand Journal of Economics, 720--737 (2006)
R. J. HART, *Interoperability Information and the Microsoft Decision*, 28 European Intellectual Property Review, 361--365 (2006)
W. KERBER, & C. SCHMIDT, *Microsoft, Refusal to License Intellectual Property Rights, and the Incentives Balance Test of the EU Commission*, (2008), presented at the EALE Annual Conference 2008
KAI-UWE KÜHN & JOHN VAN REENEN, *Interoperability and Market Foreclosure in the European Microsoft Case*. LSE Centre for Economic Performance (February, 2008)
NET LE, *Microsoft Europe and Switching Costs*, 27 World Competition, 567--594 (2004)
LAWRENCE LESSIG, *The future of ideas: the fate of the commons in a connected world*, Random house, XIII+352 (2001)
FRANÇOIS LÉVÊQUE, *Innovation, Leveraging and Essential Facilities: Interoperability Licensing in the EU Microsoft Case*, 28 World Competition, 71--91 (2005)
DAVID MCGOWAN, *Between Logic and Experience: Error Costs and United States v. Microsoft Corp*, 20 Berkeley Technology Law Journal, 1185 (2005)
A. DOUGLAS MELAMED, *Exclusionary Conduct under the Antitrust Laws: Balancing, Sacrifice, and Refusals to Deal*, 20 Berkeley Technology Law Journal, 1247 (2005)

MARIA LILLÀ MONTAGNANI, *Predatory and Exclusionary Innovation: Which Legal Standard for Software Integration in the Context of the Competition v Intellectual Property Rights Clash?*, SSRN-id804985 (working draft) (September, 2005)
FEDERICO MORANDO, *Principi tecnici ed economici per l'analisi del mercato delle piattaforme software: Il Caso Microsoft europeo (Economic and technical elements for the analysis of the market for software platforms: The European Microsoft case)*, 12 Concorrenza e Mercato, 165--241 (2004)
MASSIMO MOTTA, *Competition Policy : Theory and Practice*, (Massimo Motta ed., Cambridge University Press. 2004)
BARRY NALEBUFF, *Exclusionary Bundling*, 50 Antitrust Bulletin, 321--370 (2005)
CORMAC O'DEA, *A Look at the State of Knowledge on Bundling*, 20 Student Economic Review, 53--63 (2006)
ROBERTO PARDOLESI & ANDREA RENDA, *The European Commission's Case Against Microsoft: Fool Monti Kills Bill?*, LE Lab Working Paper No. AT-07-04 (August, 2004)
FRANCESCO PARISI, et al., *Duality in Property: Commons and Anticommons*, 25 International Review of Law and Economics, 578--591 (2005)
MARCO RICOLFI, *Is There an Antitrust Antidote Against IP Overprotection within Trips?*, 10 Marq. Intell. Prop. L. Rev., 305--367 (2006)
J. C. ROCHET & J. TIROLE, *Platform Competition in Two-Sided Markets*, 1 Journal of the European Economic Association, 990--1029 (2003)
J. C. ROCHET & J. TIROLE, *Two-Sided Markets: An Overview*, IDEI Toulouse working paper (March, 2004)
J. C. ROCHET & J. TIROLE, *Two-Sided Markets: A Progress Report*, 37 RAND Journal of Economics, 645--667 (2006)
MIKKO VÄLIMÄKI, *Software Interoperability and Intellectual Property Policy in Europe*, 3 European Review of Political Technologies, 1--11 (2005)
IVO VAN BAEL & JEAN-FRANÇOIS BELLIS, *Competition Law of the European Community*, (Kluwer Law International. 2005)
M. WHINSTON, *Tying, Foreclosure and Exclusion*, 80 American Economic Review, 857--873 (1990)
MICHAEL D. WHINSTON, *Exclusivity and Tying in U.S. v. Microsoft: What We Know, and Don't Know*, 15 The Journal of Economic Perspectives, 63--80 (2001)
BARBARA ANN WHITE, *Choosing Among Antitrust Liability Standards Under Incomplete Information: Assessments of and Aversions to the Risk of Being Wrong*, 20 Berkeley Technology Law Journal, 1173 (2005)
J. ZITTRAIN, *The Un-Microsoft Un-Remedy: Law Can Prevent the Problem that It Can't Patch Later*, 31 Connecticut Law Review, 1361 (1999)